BIBLIOTECA BASICA INFORMATICA

¡Y LLEGO EL BASIC! (I)

5



INGELEK

BIBLIOTECA BASICA INFORMATICA

¡Y LLEGO EL BASIC! (I)

5

INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Molsalve.

Colaboradores:
Angel Segado.
Patricia Mordini.
Margarita Caffaratto.
Marina Caffaratto.
Francisco Ruiz.
Jorge Juan Monsalve.
Beatriz Tercero.
Fernando Ruiz.
Casimiro Zaragoza.

Diseño:
Bravo/Lofish.

Dibujos:
José Ochoa.

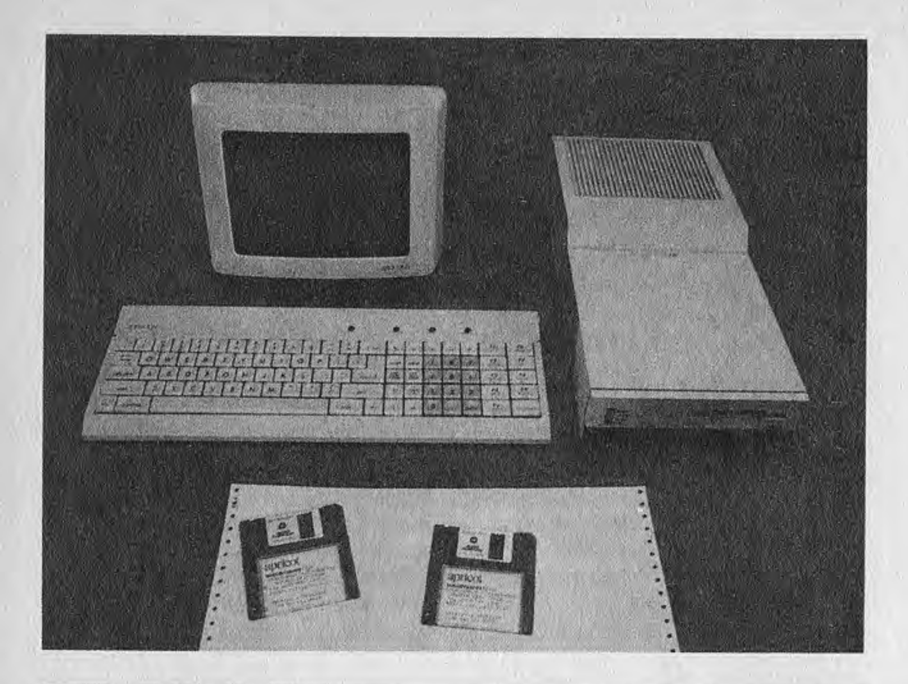
ISBN del tomo: 84-85831-36-5
ISBN de la obra: 84-85831-31-4
Fotocomposición: Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-37692-1985

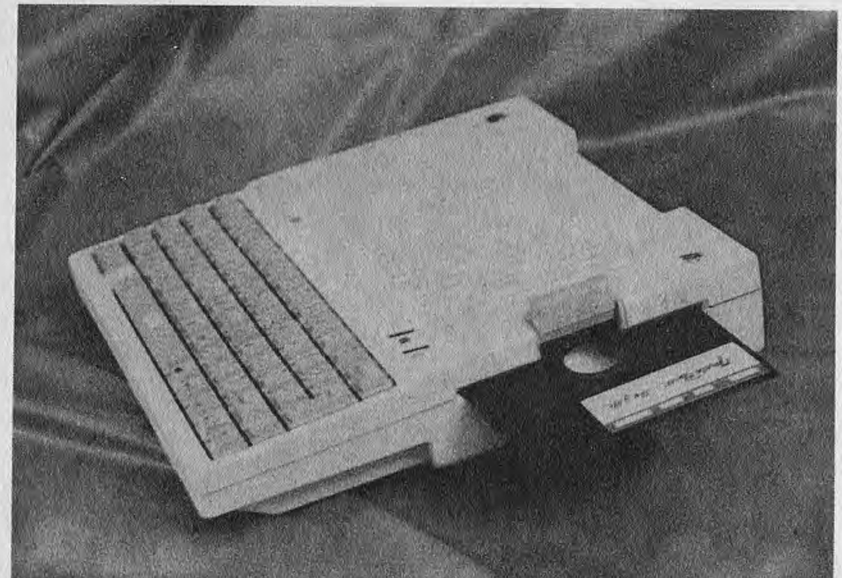
INDICE

PROLOGO

Ordenador doméstico, ordenador personal, microordenador, etc., ya son términos de uso común y forman parte de nuestras expresiones cotidianas. Los ordenadores, hasta hace muy poco dominio de pocas personas privilegiadas, han conquistado actualmente de forma clara su lugar en nuestros hogares. Algunos niños los utilizan como juguetes, otros para hacer los deberes escolares, mientras que para los adultos resultan un pasatiempo agradable, aunque cada vez con mayor frecuencia constituyen también su instrumento de trabajo.

Los ordenadores son máquinas y, como tales, realizan trabajos. Lo que los distingue claramente del resto es que este trabajo no es uno fijo e invariable, sino que se adaptan a cuanto se les solicita; y esto se debe a que los ordenadores son programables. Programar un ordenador significa enseñarle qué es lo que queremos que haga. ¿Y si no los programamos? No servirán para nada, se convertirán en unos objetos inertes e inútiles. Al comprar un ordenador podemos creer que hemos adquirido solamente componentes electrónicos tales como transistores, resistencias y circuitos integrados lo que los especialistas llaman «hardware» (vea el volumen 1 de la B. B. I.). Pero no es así, puesto que tomamos también el «software de base», esto es, programas muy



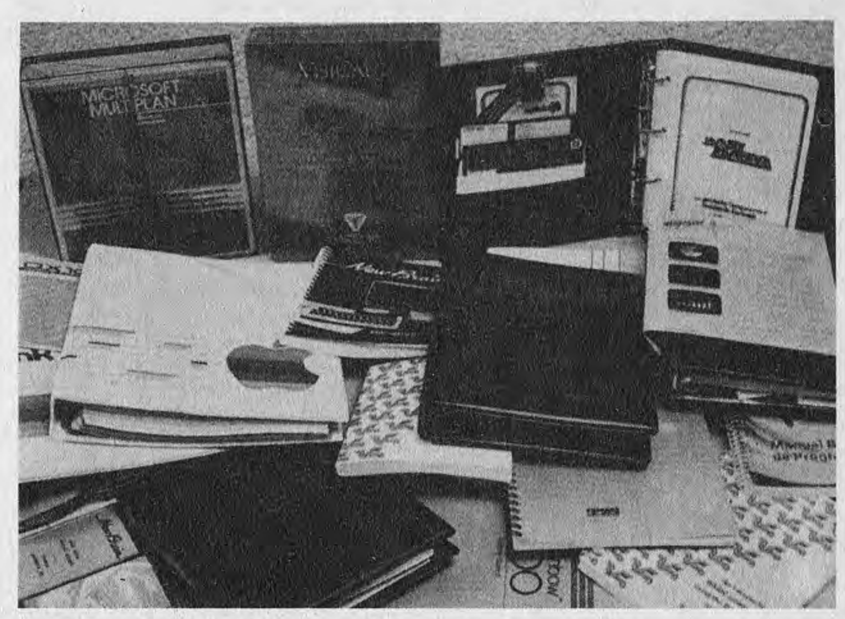


Fotografía 1.—Diversas muestras de ordenadores personales con periféricos de almacenamiento masivo exterior (arriba) o interior (abajo).

importantes que hacen funcionar el sistema. Estos programas son normalmente, en un ordenador personal, el Intérprete BASIC y otro, a primera vista ininteligible: el Sistema Operativo.

Frecuentemente además del software de base, se pueden obtener otros programas que vienen «escritos» en cintas magnéticas, en cartuchos electrónicos especiales del tamaño de una caja de cigarrillos o grabados en discos flexibles. En todos estos casos, no tenemos que hacer nada más que encender el ordenador, insertar el programa («cargarlo» como se dice en la jerga) y, luego, ejecutarlo.

Si hemos cargado un juego, probablemente tengamos que demostrar nuestra capacidad para destruir las astronaves enemigas ayudadas por un pequeño dispositivo de disparo y control, denominado «joystick», que nos hará sentir como pilotos espaciales. Si se trata de un programa didácticos de matemáticas, podremos repasar algún problema de geometría. Si, por el contrario, se trata de un programa de tipo profesional, por ejemplo de contabilidad, deberemos in-



Fotografía 2.—Manuales y libros de programación.

troducir los datos que nos pida y encender la impresora si queremos obtener resultados impresos.

Los ordenadores no ejecutan solamente programas que, escritos por otras personas, compremos; por supuesto podemos programarlos nosotros mismos, empleando las reglas de programación establecidas por el fabricante. El conjunto de estas reglas constituye lo que se denomina lenguaje de programación. Se habla de lenguaje porque se emplea para «comunicar» al ordenador lo que debe hacer, del mismo modo que cuando hablamos a alguien decimos «haz esto así». Estas frases son las denominadas «instrucciones». Muchas instrucciones juntas, ordenadas según un sentido lógico, constituyen un programa.

LENGUAJE	AÑO	SECTOR DE APLICACION
FORTRAN	1954	científico-matemático
ALGOL	1958	científico-matemático
LISP	1958	científico para aplicaciones de "inteligencia artificial"
COBOL	1959	comercial
APL	1962	científico
PL/1	1964	comercial y científico
BASIC	1964	para uso general
FORTH	1969	aplicaciones industriales, robótica
LOGO	1970	enseñanza
PASCAL	1971	científico y comercial
C	1973	aplicaciones industriales
ADA	1979	para sistemas complejos
MODULA-2	1983	científico y general

Fig. 1.—Lenguajes de programación más importantes.

Los lenguajes de programción nacieron junto con los ordenadores. Los más antiguos tienen nombres conocidos, tales como FORTRAN o COBOL. Otros son muy populares y actualmente están al alcance de todos, como es el caso BASIC, [mientras algunos son sólo conocidos por técnicos o especialistas en los campos donde su aplicación resulta ventajosa (ADA, C,...)].

Por si desea acudir a él durante la lectura del libro, le llamamos la atención sobre el apéndice B, que contiene la definición, sintaxis y ejemplos de cada una de las instrucciones del BASIC que se verán en este volumen de la B.B.I.

CAPITULO I

ENTRE LINEAS ANDA EL BASIC

Fundamentos del BASIC

El BASIC nació en el año 1964. ¿Sorprendido? En efecto, suele asociarse su nacimiento con el de los ordenadores personales (hace unos diez años), pero en realidad el BASIC tiene unos orígenes más lejanos, que se remontan casi a la prehistoria de los ordenadores. Sus hermanos mayores (FORTRAN, COBOL y otros) eran lenguajes complejos que requerían largos períodos de tiempo entre la fase de escritura del programa y su ejecución sin errores. Su empleo estaba reservado a una élite de especialistas que trabajaban con grandes ordenadores.

Dos profesores americanos, Kemeny y Kurtz, pensaron que, para facilitar el empleo de los ordenadores a estudiantes y profanos, era preciso simplificar bastantes cosas. Su ideal era poder sentarse delante del ordenador, programarlo directamente, ejecutar el programa y recibir los resultados (... precisamente lo que hacemos actualmente con los ordenadores personales). Kemeny y Kurtz lograron sus propósitos y así nació el BASIC. La palabra BASIC es la abreviatura de «Beginner's All-purpose Symbolic Instruction Code» que significa «Lenguaje de programación para principiantes.»

Es este carácter de «inmediatez», logrado gracias a la

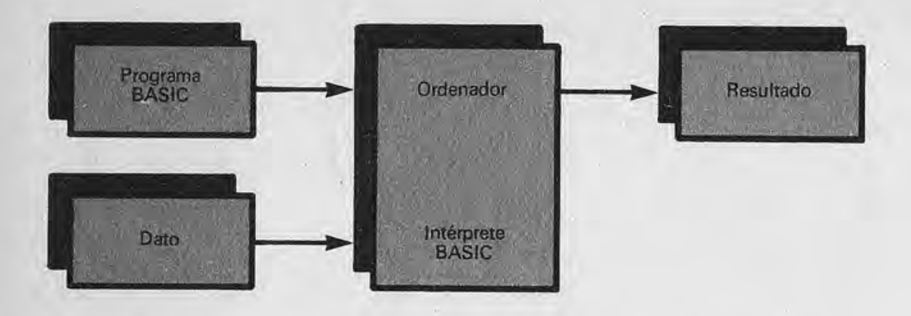


Fig. 1.—La traducción del programa al código interno del ordenador se realiza por medio del Intérprete BASIC.

acción del Intérprete BASIC, lo que ha otorgado al BASIC la inmensa popularidad de que ahora goza.

Su gran difusión se produjo, sin embargo, (y de ahí los equívocos) cuando fue adoptado como lenguaje principal por los ordenadores personales. Es evidente que fue un «matrimonio» de intereses: se produjo porque ambos, el ordenador personal y el BASIC, pretendían las mismas objetivos: resultar sencillos y fáciles de utilizar. Con toda seguridad, el BASIC es en la actualidad el lenguaje de programación más conocido unviersalmente.

Como mencionamos antes, uno de los aspectos fundamentales del BASIC «culpable» de su facilidad de uso, es que se basa en un Intérprete. Como se observa en la Figura 1, el Intérprete es un programa particular (escrito en lenguaje máquina) que admite como «entradas», además de los datos, las instrucciones del programa BASIC del usuario, traduciéndolas a un código «conocido» y admitido por el ordenador. Y esto lo hace instrucción a instrucción. Así, aunque en un programa modifiquemos alguna instrucción podemos ejecutarlo sin problemas mientras que en un lenguaje como el FORTRAN o el PASCAL cualquier pequeño cambio (aunque sea tan solo para añadir un «punto» que faltaba) supone someter de nuevo todo el progrma a un proceso, previo a su ejecución, denominando compilación y que lleva bastante tiempo.

Esta es una importante ventaja de los lenguajes «interpretados» (como el BASIC) respecto a los «compilados», en los cuales la «traducción» del programa completo se ha de realizar previamente. La sencillez que se deriva de tal circunstancia tiene su contrapartida en una menor velocidad de ejecución: el Intérprete tiene que traducir las instrucciones cada vez que las ejecuta, mientras que con los compiladores la traducción se hace una sola vez.

La enorme proliferación del uso del BASIC ha traido como consecuencia lógica, el nacimiento de muchos dialectos, de forma análoga a lo que sucede con las lenguas habladas. Cuando a partir de una, original, nacen muchas variantes de una misma lengua, pueden llegar a producir confusiones en su aprendizaje. Así sucedió también en el caso del BASIC. Las principales y más importantes instrucciones se mantienen iguales de una versión a otra, pero como cada fabricante ha añadido a su ordenador peculiaridades intrínsecas a su marca, resulta prácticamente imposible que un programa escrito para un ordenador concreto pueda ejecutarse directamente en otro: una coma colocada en diferente lugar o suprimida basta para que la máquina se niegue a «acatar» y ejecutar un programa. Así pues, es imaginable lo que puede suceder si cada ordenador modifica las reglas de las instrucciones BASIC.

En contra de esta tendencia «dispersora» ha surgido con fuerza en los últimos años un deseo de «normalización», que se está viendo materializado con los equipos MSX. Sin embargo, este nuevo estándar no ha sido todavía universalmente reconocido. Por tanto hoy en día hablar de BASIC, en general, quiere decir describir la estructura del lenguaje, sus instrucciones principales, los conceptos comunes a todos los dialectos y advertir cuando existan divergencias importantes.

Para adaptar después un programa en BASIC a un ordenador específico, se necesita tener la paciencia de amoldarlo a algunas de sus instrucciones particulares. Nosotros, en este volumen de la B. B. I., hablaremos del BASIC en los términos más generales posibles. Es oportuno advertir, no obstante, que las principales diferencias entre las distintas versiones del BASIC se encuentran en las complejas instrucciones que controlan los gráficos o la gestión de archivos, las cuales se presentarán en un próximo libro (segundo de los dedicados específicamente al BASIC)



Fotografía 2.—Listado de un programado BASIC tal como aparece por pantalla o impresora.

todas las líneas del programa) de cualquier programa, obtendremos la impresión de que está escrito en inglés. Lamentablemente, sobre esta cuestión no hay nada que hacer. Todo el mundo técnico, científico y de negocios, ha adaptado ya este idioma, por lo que cuando surje un nuevo lenguaje de programación utiliza siempre palabras inglesas, consideradas más «internacionales» que las demás. Como confirmación de tal circunstancia basta recordar que uno de los más recientes lenguajes de programación, el PASCAL, fue desarrollado en Suiza pero utiliza términos ingleses.

Un programa escrito en BASIC se reconoce inmediatamente por la especial característica de tener todas sus líneas numeradas. El incrementar los números de línea de 10 en 10, como haremos normalmente en nuestros ejemplos, es sólo por una cuestión práctica: dejar espacio para la inserción de nuevas instrucciones con un número de línea intermedio entre los de dos instrucciones ya introducidas. Aunque no lo parezca esta medida puede evitarnos muchos disgustos, especialmente en las versiones del BASIC que no admiten la «renumeración» de las líneas.

Numerar las líneas de un programa en BASIC es muy sencillo: escribir al comienzo de la línea un número positivo

entero, es decir, un número sin coma ni parte decimal, y tener en cuenta algunas reglas:

- Dos líneas no pueden tener el mismo número. En caso contrario el ordenador «recuerda» solamente la que se escribió en último lugar y olvida la anterior.
- Una línea no puede estar constituida solamente por el número, sino que debe contener una instrucción efectiva por lo menos. Si escribe solamente el número, el ordenador ignorará esa línea.
- Para borrar (anular) una línea basta volver a escribir su número y nada más. El ordenador sustituirá entonces la línea antigua por la nueva (primera regla), vacía y, por consiguiente la anula (segunda regla).
- El número de línea puede ser cualquier número entero positivo, el cero (0) no se utiliza casi nunca y el número máximo es una característica del propio ordenador concreto que utilicemos. Para saber cuál es su valor deberemos consultar el manual correspondiente del aparato. En cualquier caso suele ser muy alto y, por ello, suficiente para escribir los programas. Si, por ejemplo, este número fuera 32767, significaría que se podrían escribir 32.768 líneas de programa (contando la número 0).
- Las líneas de un programa representan una secuencia lógica de órdenes impartidas al ordenador y, por este motivo, la numeración de las líneas debe tener la misma progresión lógica. Los números de línea no deben ser necesariamente correlativos, como lo son «23, 24, 25, 26...», sino solamente respetar el orden adecuado. El ordenador leerá las instrucciones en orden ascendente (desde el menor al mayor número de línea usado).

Con referencia al Intérprete debemos insistir en que para incorporar cualquier línea al programa, las instrucciones de que conste deberán ir precedidas por el número de línea. Al hacer esto evitamos que se interpreten las instrucciones como de «inmediata» ejecución, bloquándose esta hasta que demos el comando RUN, momento en el cual entrará en acción el Intérprete para «traducir» todas las veces que sean necesarias las instrucciones.

Si, por el contrario, no tecleamos los números de línea, la ejecución del comando individual será inmediata (siem-

pre a cargo del Intérprete). Precisamente por eso, cuando deseamos usar las funciones del sistema, tales como LIST (para proporcionar el listado de las instrucciones del programa actualmente en memoria), LOAD y SAVE (para cargar/grabar un programa desde/en un soporte magnético), RUN (ya visto), etc., no les anteponemos un número de línea.

Si deseamos escribir un programa que ejecute consecutivamente tres instrucciones (primera, segunda y tercera) la redacción

30 Primera instrucción.
45 Segunda instrucción.
120 Tercera instrucción.

Es correcta y, en cambio, no lo sería:

30 Segunda instrucción.
45 Primera instrucción.
120 Tercera instrucción.

Porque el ordenador ejecutará primero la instrucción de número más bajo, es decir la 30, que queremos se ejecute en segundo lugar.

- Cuando escribimos un programa, es decir, cuando lo introducimos a través del teclado del ordenador, podemos escribir primero una línea con un número más alto y luego otra con número más bajo, es decir, no hace falta introducir las líneas por el orden de sus respectivos números de línea.

La explicación de lo anterior es muy simple: el ordenador ordena las líneas de forma automática. Por consiguiente, se puede escribir un programa de este modo:

60 Tercera instrucción.
40 Segunda instrucción.
20 Primera instrucción.

Porque el aparato las ordenará de forma automática como sigue:

20 Primera instrucción.
40 Segunda instrucción.
60 Tercera instrucción.

- Si en cualquier momento comprobamos que hemos olvidado una instrucción, podemos introducirla sin ningún problema asignándola el número de línea adecuado.

Supongamos, por ejemplo, que hubiéramos escrito el programa numerando las líneas de 10 en 10, como es recomendable hacer. Si en ese momento nos percatáramos de que se nos ha olvidado una (o más) podríamos teclear la nueva línea con un número intermedio. Por ejemplo, si hubiéramos escrito:

10 Primera instrucción.
20 Segunda instrucción.
30 Tercera instrucción.
40 Cuarta instrucción.

Y quisiéramos introducir una instrucción entre la tercera y la cuarta, nos bastaría teclear:

35 Nueva instrucción/tercera bis.

El ordenador la pondrá automáticamente en orden como sigue:

10 Primera instrucción.
20 Segunda instrucción.
30 Tercera instrucción.
35 Nueva instrucción/tercera bis.
40 Cuarta instrucción.

CAPITULO II

BLOQUES Y VARIABLES: DEFINIENDO UN PROGRAMA

Un ordenador es una máquina que, por sí sola, no sabe realizar ninguna acción; necesita un programa que la transforme en una máquina viva. Para comunicar al ordenador este programa es necesario utilizar un lenguaje, un «lenguaje de programación». El BASIC es uno de los más importantes porque es muy fácil de aprender y por haber sido adoptado por todos los ordenadores personales.

Insistimos en que la base de todo sigue siendo el hecho de que el ordenador no inventa nada y de que el programa debe ser concebido y escrito por nosotros, los usuarios. Si no sabemos cómo resolver un problema, no podemos pretender que el ordenador lo resuelva por sí solo.

Antes de programar un ordenador debemos tener las ideas muy claras y saber con exactitud qué queremos pedir a la máquina y cómo hacer que esta pueda ofrecérnoslo.

Si, por ejemplo, un programador experto en leyes tributarias describe con «pelos y señales» a un ordenador cómo completar los calculados del modelo 740 (es decir, introduce programa para realizar todas las operaciones correspondientes), ya no tendrá necesidad de repetir las cuentas a mano para cada formulario: una vez introducidos los datos, el ordenador se encargará de todo.

¿Qué es lo que quiere decir «explicar al ordenador con pelos y señales»? Significa definir, sin omisión ni ambigüe-

dad alguna, la forma en que precisa comportarse en todas las posibles situaciones que se puedan presentar durante el trabajo; en otras palabras: definir un algortimo (en el ejemplo anterior para rellenar el modelo 740).

Solamente después de haber definido un algoritmo que tenga en cuenta todos los casos posibles, se podrá «traducir» éste al lenguaje BASIC y escribirlo en el ordenador.

Aunque le «suene a chino» eso de los algoritmos, conocemos muchos y los utilizamos cada día sin reconocerlos como tales. Son algoritmos las reglas aprendidas en la escuela para efectuar las sumas o las divisiones, el procedimiento que seguimos para arrancar el automóvil ¡e incluso el método que empleamos para hacer café!

Los algoritmos

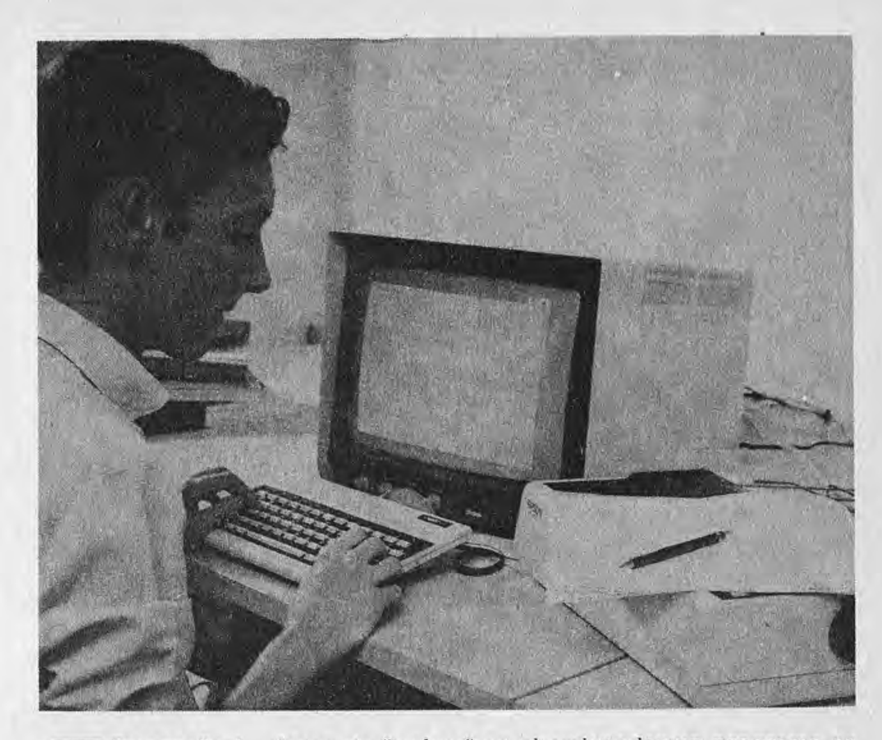
La palabra algoritmo se deriva del nombre del matemático y astrónomo áraba Mohammed al-Khuwarizmi, que vivió antes del año 850.

Entre sus diversas obras estaba una profunda y completa recopilación de los sistemas de numeración entonces utilizados en la India. El sistema indio, que representa a los números en forma posicional, fue evolucionando hasta convertirse en nuestro sistema de numeración actual. Las cifras 0, 1, 2, 3 etc., que llamamos «arábigas» para distinguirlas de las romanas (I, V, X, C etc), son en realidad de origen indio. Este error «histórico» se debe al hecho de que cuando las obras de Mohammed al-Khuwarizmi fueron traducidas al latín se creyó que todo era árabe como el texto, incluido el sistema de numeración presentado, que fue denominado «algorismo», término del cual se derivó posteriormente algoritmo.

Gracias, pues, a este matemático árabe, podemos realizar actualmente la mayor parte de las operaciones aritméticas con números.

La palabra «algoritmo» acabó por significar cualquier conjunto de reglas, primero para un cálculo matemático y luego para un problema cualquiera.

A propósito de Mohammed al-Khuwarizmi, le somos deudores no solamente de los números «arábigos», sino también de las bases de una rama fundamental de las matemáticas, que conocemos como álgebra. La palabra «álgebra» se deriva del título de la obra más importante de Mohammed al-Khuwarizmi: «Al-jabr wà Imugabalh» (y no es broma), Del término «al-jabr» se deriva precisamente la palabra «álgebra».



Fotografía 1.—Antes de "teclear" en el ordenador un programa es conveniente realizar un estudio previo del mismo que evite improvisaciones y sorpresas.

Cuando deseamos escribir un programa en BASIC que nos resuelva un problema, no es conveniente sentarse inmediatamente delante del teclado y comenzar a introducir las sentencias (o líneas del programa). Los modernos ordenadores personales son tan cómodos de utilizar y tan atractivos en su interactividad (diálogo del «tipo pregunta-respuesta») que suelen inducir a cometer este error con facilidad.

¿Qué sucede cuando se escribe un programa «de golpe y porrazo» sin ponerse primero a considerar a fondo to-

dos los aspectos del problema? Al principio, todo parece fácil. Luego, se descubre que el programa presenta problemas en alguna situación que no estaba prevista. Esto no tiene «nada de malo», puesto que bastará una pequeña corrección, pero esta última, a su vez, puede hacer surgir un incoveniente en otro caso: tendremos que realizar otro pequeño retoque y así sucesivamente...

Cuando al final de todo el proceso anterior el programa funcione a la perfección (si es que lo hace), descubriremos haber empleado mucho más tiempo del que hubiera sido necesario, partiendo de un estudio «serio» del problema, antes de comenzar a escribirlo. Pero este no es el único incoveniente; hay otros mucho más graves:

- No tendremos la certeza del funcionamiento correcto en cualquier condición.
- Será difícil comprender cómo funciona el programa, particularmente después de transcurrido algún tiempo desde la escritura, pues los sucesivos cambios habrán deshecho una posible estructura inicial.
- En consecuencia, es también más difícil modificarlo en caso de necesidad.
- El programa será «más o menos» comprensible solamente para el autor, imposibilitando su fácil intercambio.

Para evitar todo lo anterior basta introducir algún comentario (con la instrucción REM que ya veremos) en el propio programa. El lenguaje BASIC no es fácilmente legible ni comprensible, y resulta complicado cuando menos entender las intenciones del programador, a partir de la simple lectura de las instrucciones del programa.

Es, pues, necesario que el programador describa de forma clara la estructura y el funcionamiento del programa, así como su funcionamiento en cualquier condición posible.

¿Cómo se puede llevar esto a la práctica?:

Escribiendo un algoritmo que describa por completo el programa. Al realizar este algoritmo, antes de ponernos con la escritura del programa, se obtienen las notables ventajas siguientes:

- Los problemas se abordan con la lógica normal y se describen en nuestro idioma, y no en BASIC (siempre más «lejano» a nosotros).

- La escritura del programa se convierte parácticamente en una traducción del algoritmo al BASIC.
- El funcionamiento del programa es perfectamente comprensible incluso para otras personas.
- Es mucho más fácil estudiar eventuales modificaciones en el algoritmo e insertarlas, luego, en el programa.

Conviene, pues, antes de iniciar la escritura de un programa, desarrollar el algoritmo resolutivo del problema.

Diagramas de flujo

Un algoritmo, escrito con palabras, presenta un inconveniente: la dificultad de seguir el flujo lógico del programa (es decir, el recorrido). Dicho de otro modo, no es fácil hacerse una idea global para determinar de un vistazo qué instrucciones se ejecutan y cúales no, qué hacen y sobre la base de qué condiciones.

Para resolver este problema se han desarrollado varias técnicas. La más adaptada al BASIC es una técnica gráfica denominada de diagramas de bloques o diagramas de flujo (flowchart).

En las Figuras 1 y 2 se indican los símbolos más importantes de los que hace uso. En la primera se presentan los que sirven para ilustrar los programas (y los algoritmos), mientras que la Figura 2 comprende los símbolos utilizados para referirse a los soportes físicos de los datos y a las unidades periféricas de entrada/salida (E/S).

En un diagrama de flujo aparecen muchos de estos símbolos (presentados en forma de rectángulos, rombos y otras figuras geométricas) unidos mediante líneas. Las líneas indican la conexión lógica entre una operación y la sucesiva. Los bloques (las figuras geométricas) representan los diversos tipos de instrucciones ejecutables. Dentro de cada bloque se describen la operación u operaciones que simboliza.

Veamos inmediatamente un ejemplo. En la Figura 3 se muestra el diagrama de bloques correspondiente al algoritmo «Cómo hacerse un café» que describimos con los pasos siguientes:

1) Coger la cafetera.
2) Coger el bote del café.

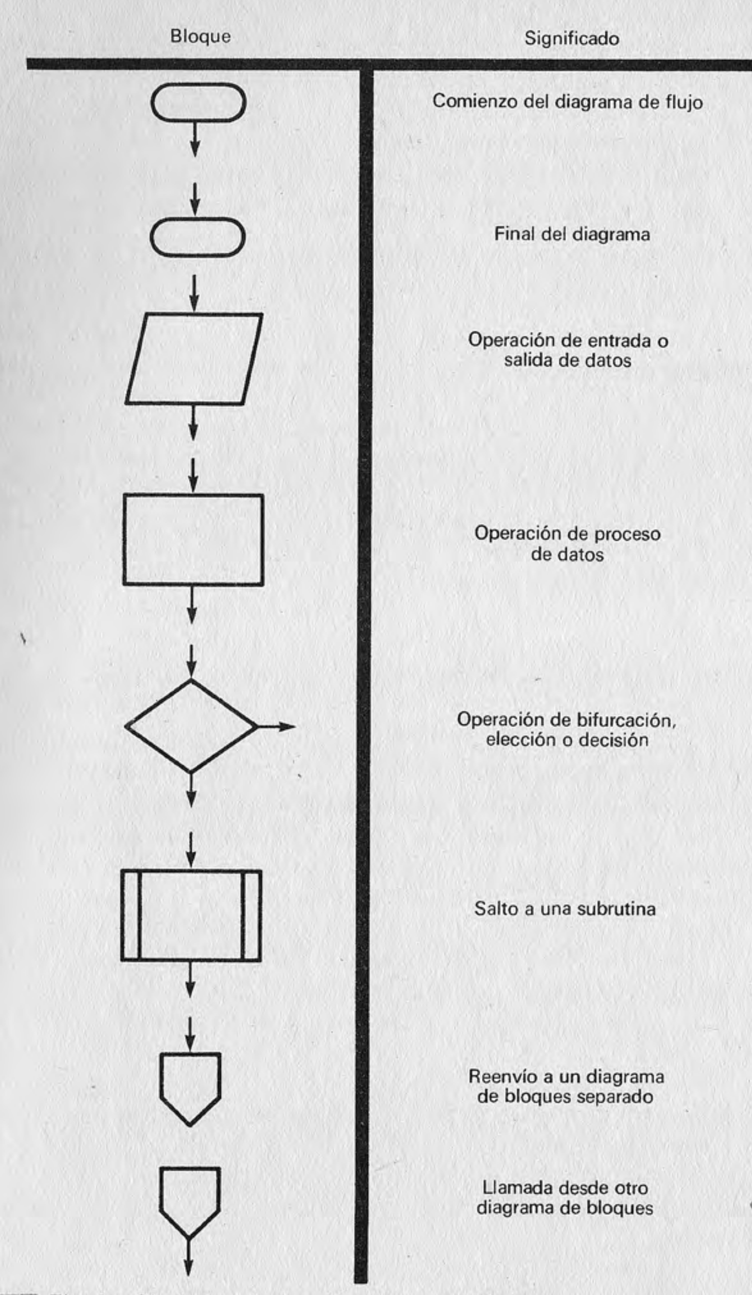


Fig. 1.—Símbolos de los diagramas de flujo.

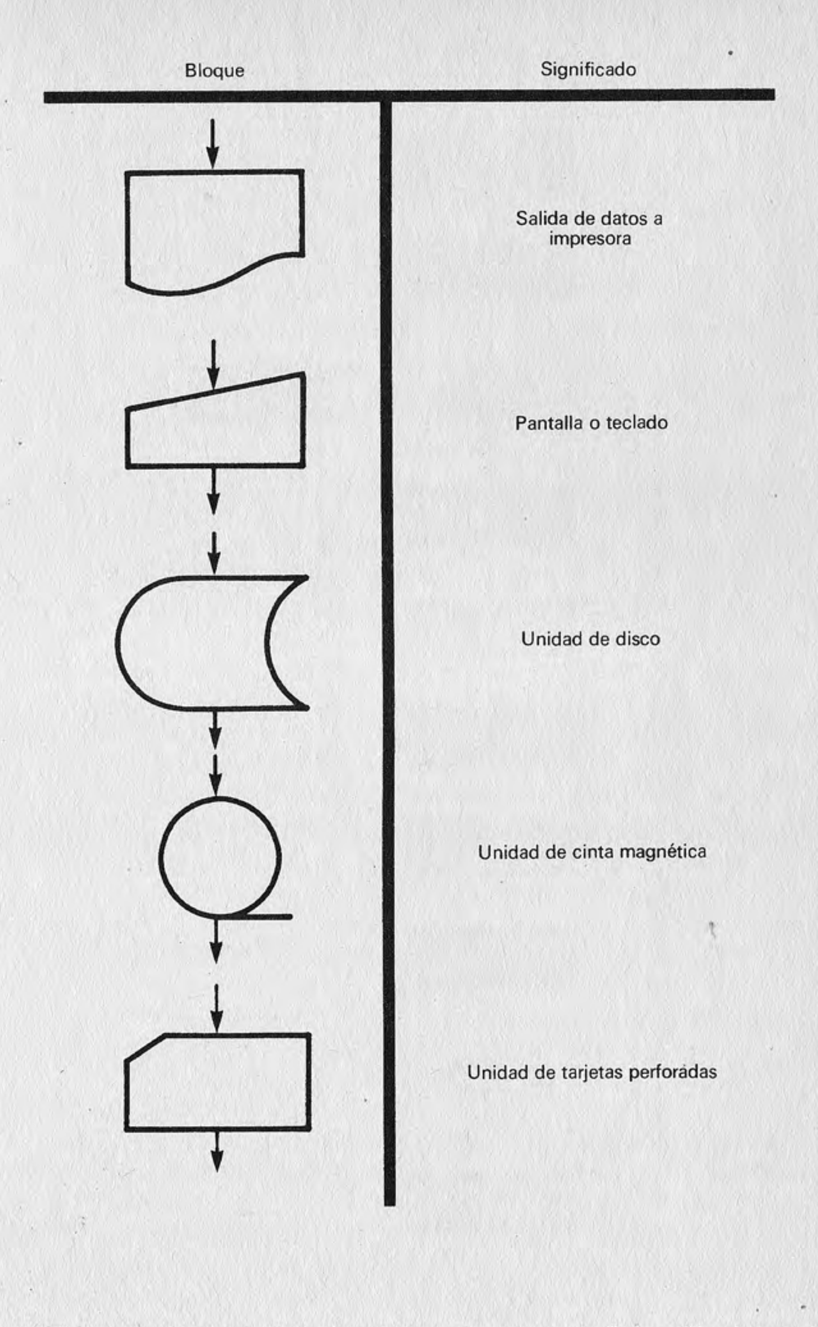


Fig. 2.—Algunos símbolos de los diagramas de flujo utilizados para indicar las unidades periféricas.

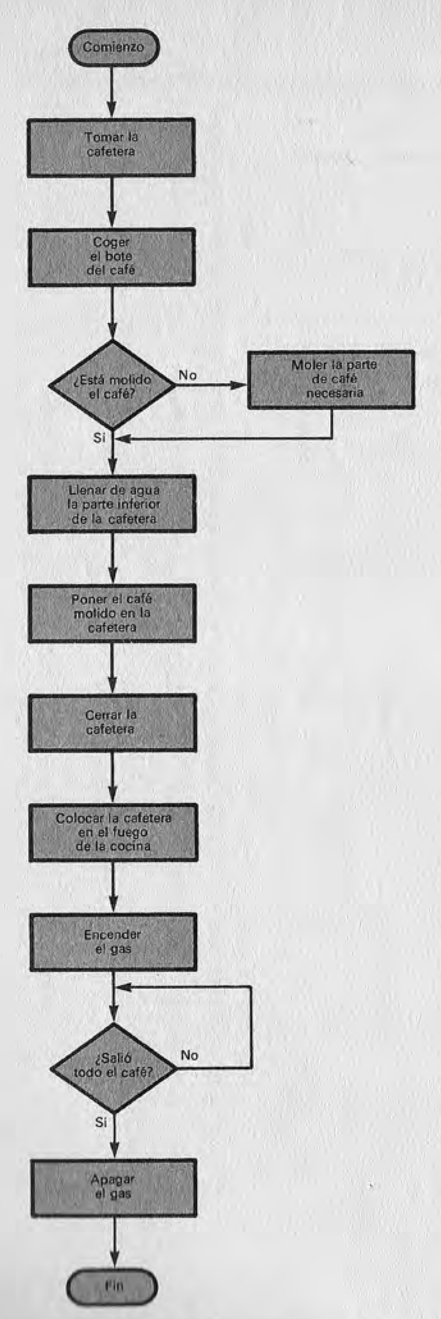


Fig. 3.—Diagrama de bloques del programa "Cómo hacerse un café".

3) Si el café está molido, saltar el resto de esta instrucción y proseguir con la instrucción 4. Si no lo está, molerlo (y, una vez molido, pasar al punto 4).

4) Llenar de agua la parte baja de la cafetera.

5) Llenar de café molido la cazoleta correspondiente.

6) Cerrar la cafetera.

7) Ponerla sobre el fuego de la cocina.

8) Encender el gas.

9) Comprobar si el café ha salido en su totalidad; si no fuera así, esperar hasta que ocurra.

10) Apagar el gas.

11) Final: el café está listo para servir.

Un diagrama de flujo se lee a partir del círculo que indica el comienzo del algoritmo (en los diagramas de flujo utilizaremos por comodidad una «pseudoelipse» en su lugar) y siguiendo las líneas, que representan la secuencia lógica en la que se ejecutarán las operaciones. Las líneas se recorrerán desde arriba hacia abajo y desde la izquierda a la derecha, salvo si existe una flecha en sentido contrario.

Partamos, pues, de la pseudoelipse inicial (en la parte superior) y sigamos la línea que desciende. Encontraremos un rectángulo correspondiente al punto 1 del algoritmo.

Un rectángulo significa: ejecutar la operación descrita (un proceso o cálculo cualquiera) y, luego, proseguir a lo largo de la línea.

Proseguimos, pues, y encontramos otro rectángulo (punto 2). Ejecutamos también esta operación y continuamos, encontrándonos, esta vez con un símbolo diferente: un rombo.

Un rombo representa una operación de elección (decisión) y, a diferencia con el rectángulo tiene más de una línea de salida. El camino a seguir será el correspondiente a la decisión tomada. Nuestro rombo corresponde al punto 3 de este programa: ¿está molido el café? Sigamos la línea correspondiente a cada respuesta. Si la respuesta es afirmativa (SI), pasamos al punto 4, y si la respuesta en negativa (NO) pasaremos al rectángulo de «moler el café», después de lo cual la línea se encuentra nuevamente con la que corresponde a la respuesta afirmativa, lo que indica, de forma gráfica, que después de este pequeño bucle llegamos también al punto 4.

Lo que acabamos de ver es, en definitiva, un ejemplo de ejecución condicional de una instrucción: el café se muele si, solamente si, la respuesta a la pregunta anterior (¿está molido ya?) era NO.

Resulta fácil, en este punto, comprobar cómo el resto del diagrama corresponde, punto por punto, al algoritmo del café, hasta su conclusión, simbolizada, como el principio, por el dibujo de una pseudoelipse.

En el punto 9 (un rombo y, por consiguiente, una elección) se nos pregunta si el café ha salido en su totalidad. Si la respuesta es afirmativa (SI), proseguirá el diagrama (y el programa) mientras que de no ser así, una línea nos hace retornar al comienzo del propio punto, esperando que se cumpla la condicción de salida. Una estructura de este tipo se denomina bucle de espera, por cuanto se mantiene a la espera de que sea cierta una condición antes de proseguir.

Símbolos fundamentales de los diagramas de flujo

Volvamos a la Figura 1, en la que habíamos indicado los símbolos principales de los diagramas, según uno de los sistemas más utilizados. Lamentablemente, no existe una normalización, es decir, un sistema universalmente aceptado.

El bloque fundamental es el rectángulo, que indica una acción (un proceso o un tratamiento) a desarrollar. Por ejemplo, multiplicar dos números o determinar el menor entre una serie de valores. Con referencia a una posible clasificación de las instrucciones que veremos más adelante, el rectángulo se emplea para las instrucciones declarativas y para las ejecutivas de cálculo.

Una variante del rectángulo es el rectángulo con dos barras verticales en los lados, que se utiliza a menudo para indicar la llamada una subrutina (parte del programa que desarrolla una determinada tarea y que, al acabarla, vuelve al punto desde el cual se la llamó). La subrutina se describiría, a su vez, con un diagrama de bloques separado.

El rombo indica una elección entre dos (o más raramente, tres) salidas. Se suele plantear una pregunta (condición) y si la respuesta es afirmativa (SI) el programa prosigue por una parte y si la respuesta es negativa (NO), proseguirá por

la otra. Este símbolo representa a las instrucciones de salto condicional o de ejecución condicional.

El paralelogramo indica una operación genérica de entrada o salida de datos (E/S). En diagramas de otro tipo se utilizan símbolos diferentes según el dispositivo periférico empleado (lector de fichas, unidad de disco, de cinta, etc.). Los símbolos empleados en este caso (Figura 2) recuerdan, de una manera estilizada, el periférico que representan. En los diagramas de bloques que representan programas se emplea siempre un mismo símbolo porque se hace referencia a la instrucción de E/S más que a la propia máquina. Utilizaremos el paralelogramo para indicar todas las operaciones de entrada y de salida.

Para indicar el comienzo y el final de un algoritmo se emplean círculos (o pseudoelipses) con las palabras «Comienzo» y «Fin» (o bien, en inglés, «Start» y «End»).

Las variables

En los primeros ejemplos de programas que vimos anteriormente, encontramos ya variables. Las variables son como una especie de cajas en las que se introducen los datos sobre los que trabaja el programa. Desde el punto de vista físico, dichas cajas son celdillas, o grupos de celdillas, de la memoria central del ordenador.

El concepto de variable de los lenguajes de programación no es muy diferente del aprendido en la escuela durante las lecciones de matemáticas. Una variable es un nombre simbólico usado para indicar una magnitud a la que se puede asignar un valor efectivo.

Un sencillo ejemplo puede aclarar algunas dudas. Supongamos que tenemos que rellenar un impreso, o formulario, para la solicitud de un certificado. Casi todos estos impresos comienzan pidiendo los datos de filiación tales como: apellidos, nombre, año de nacimiento, etc. Junto a cada uno de estos datos hay una casilla en la que tenemos que escribir nuestros datos personales:

Apellidos .... Zorrilla Vizcaino
Nombre .... Ignacio
Año de nacimiento 1974

«Apellidos,» «Nombre» y «Año de nacimiento» serían variables del impreso a las que se atribuirán valores efectivos en nuestro caso, Zorrilla Vizcaino, Ignacio y 1974. Todos los impresos llevan indicadas las mismas variables, a las que cada uno «asigna» sus propios datos. En el lenguaje BASIC, estas variables podrían llamarse: APELLIDO $, NOMBRE$ y AÑO.

El signo del dólar ($) que hemos colocado al final de las dos primeras variables indica en BASIC que se trata de «variables alfanuméricas» (o de cadena de caracteres). La falta de este signo (AÑO) indica que se trata de una variable numérica.

Las variables aprendidas en la escuela se referían solamente a números, mientras que las que utilizamos en los ordenadores tales como APELLIDO$ y NOMBRE$, representan un concepto más amplio.

En BASIC existen dos clases de variables: las variables numéricas y las alfanuméricas o de cadena.

Las variables numéricas se refieren a los números y con ellas se pueden ejecutar todas las operaciones matemáticas.

Las variables alfanuméricas tratan las cadenas de caracteres y con ellas se pueden efectuar solamente algunas operaciones sencillas, tales como unir dos cadenas, fraccionar una o incluir una cadena más corta en otra más larga.

En la figura 4 podemos ver variables de ambos tipos y sus distintos contenidos (o valores): en el caso de las numéricas un número (A=27) y en el de los alfanuméricas una cadena de caracteres (A$=CASA).

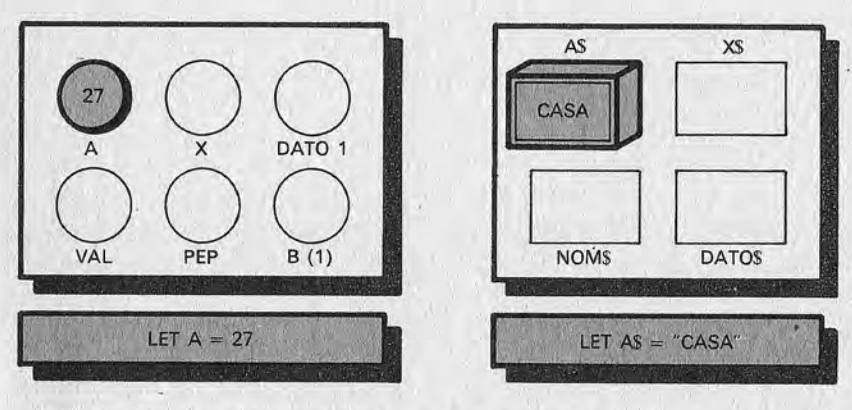


Fig. 4.—Variables numéricas y alfanuméricas (o de cadena).

Los nombres de las variables pueden elegirse de forma arbitratria en un programa, pero tienen que satisfacer algunas reglas precisas. La regla más importante se refiere al carácter inicial, que debe ser siempre una letra del alfabeto (de la A a la Z), mientras que los demás caracteres pueden ser tanto letras como cifras o incluso algunos caracteres especiales como el $, con el que debe acabar el nombre de cualquier variable alfanumérica.

Veamos algunos ejemplos:

A
AL
PIPPO
W345

Se trata de cuatro nombres correctos de variables (porque comienzan con una letra). Los dos nombres, PIPPO y W345, pueden parecer demasiado largos pero, en realidad, algunos ordenadores admiten nombres con una longitud de hasta 40 caracteres.

En el lenguaje BASIC, las variables pueden ser de diverso tipo según la magnitud que representan. El tipo de variable se suele reconocer añadiendo un carácter especial como sufijo al final del nombre de la variable. El sufijo más importante es el signo del dólar ($) que identifica a las variables alfanuméricas. Por ejemplo:

B$
MARIPOSA$
D2$

Se trata de tres nombres de variables de cadena que se reconocen inmediatamente por la presencia del signo del dólar ($). Su lectura sería: B-dólar, MARIPOSA-dólar y D2-dólar.

También las variables numéricas se distinguen en diversos tipos según la clase de número que representan:

- Las variables enteras contienen números enteros, sin parte fraccionaria o decimal, comprendidos entre —32768 y

32767. Algunos ordenadores utilizan el carácter del % para representar a estas variables (como A% o B2%).

- Las variables reales de simple precisión, es decir, las que tienen coma (128349,57). Son las variables más comunes y no hace falta ningún sufijo para distinguirlas aunque se puede emplear el carácter del signo de exclamación (!). Contienen números con seis o nueve cifras significativas.
- Las variables reales de doble precisión (utilizadas solamente por los ordenadores personales más grandes) pueden «trabajar» con números de 14 cifras y son identificables por el sufijo denominado, en la jerga, «cancela» (o sostenido en las aplicaciones de audio): #

Aunque los símbolos %, ! y # se usan, efectivamente, para determinar el tipo de la variable hay algunos ordenadores (los MSX, el Spectravideo o el IBM-PC) que dan al usuario la posibilidad de definir como pertenecientes a una de las 3 clases todas las variables cuyo nombre comience por una letra determinada. Por ejemplo:

10 DEFINIT I-N define de tipo entero todas las variables que comiencen por cualquier letra entre la I y la N.

Las equivalentes de simple y doble precisión son: DEFSMG y DEFDBL respectivamente.

CAPITULO III

INSTRUCCIONES DE ENTRADA, DE SALIDA Y DECLARATIVAS (LOS ARRAYS)

Las palabras reservadas

Cada lenguaje de programación está constituido por un conjunto bien determinado de instrucciones, tales como en BASIC: PRINT, GOTO, READ, INPUT o REM. Si se consulta un manual de BASIC de los que suministran con los ordenadores, nos percataremos inmediatamente de que las instrucciones son varias decenas y, a veces, superan incluso el centenar. No podemos indicar cuántas son con exactitud porque su número varía de un ordenador a otro según el dialecto de BASIC utilizado.

En la Figura 1 hemos indicado las "palabras reservadas" del ordenador IBM PC, pero cada ordenador tiene las suyas propias. Estas palabras reservadas son aquellas a las que el ordenador atribuye un significado particular; en la práctica, a cada una de ellas corresponde una instrucción en BASIC. El ordenador IBM PC tiene 160 palabras reservadas, algunas de las cuales son comunes a muchos a otros ordenadores. Hablar de BASIC, en general, supone referirse, precisamente, a este conjunto "central" de instrucciones.

Reducido, pues, al máximo el campo de tratamiento de las instrucciones del lenguaje BASIC, resulta fácil observar cómo se pueden clasificar en varios grupos, cada uno de los cuales desarrolla una función particular.

ABS	EOF	LPRINT	RIGHT$
AND	EQV	LSET	RND
ASC	ERASE	MERGE	RSET
ATN	ERL	MID$	RUN
AUTO	ERR	MKD$	SAVE
BEEP	ERROR	MKI$	SCREEN
BLOAD	EXP	MKS$	SGN
BSAVE	FIELD	MOD	SIN
CALL	FILES	MOTOR	SOUND
CDBL	FIX	NAME	SPACE$
CHAIN	FN*xxxxxxxx*	NEW	SPC(
CHR$	FOR	NEXT	SQR
CINT	FRE	NOT	STEP
CIRCLE	GET	OCT$	STICK
CLEAR	GOSUB	OFF	STOP
CLOSE	GOTO	ON	STR$
CLS	HEX$	OPEN	STRIG
COLOR	IF	OPTION	STRING$
COM	IMP	OR	SWAP
COMMON	INKEY$	OUT	SYSTEM
CONT	INP	PAINT	TAB(
COS	INPUT	PEEK	TAN
CSNG	INPUT#	PEN	THEN
CSRLIN	INPUT$	PLAY	TIME$
CVD	INSTR	POINT	TO
CVI	INT	POKE	TROFF
CVS	KEY	POS	TRON
DATA	KILL	PRESET	USING
DATE$	LEFT$	PRINT	USR
DEF	LEN	PRINT#	VAL
DEFDBL	LET	PSET	VARPTR
DEFINT	LINE	PUT	VARPTR$
DEFSNG	LIST	RANDOMIZE	WAIT
DEFSTR	LLIST	READ	WEND
DELETE	LOAD	REM	WHILE
DIM	LOC	RENUM	WIDTH
DRAW	LOCATE	RESET	WRITE
EDIT	LOF	RESTORE	WRITE#
ELSE	LOG	RESUME	XOR
END	LPOS	RETURN	

Fig. 1.—Las 160 "palabras reservadas" del ordenador IBM PC. Cada una de ellas constituye, en la práctica, una instrucción del lenguaje BASIC de este ordenador.

Por ejemplo, PRINT transfiere datos desde la memoria central hacia un periférico, mientras que GOTO no efectúa ninguna operación con los datos, sino que cambia el orden de ejecución de las instrucciones.

Prosiguiendo con este examen, se observa cómo es posible reconocer cuatro familias, o categorías principales, de instrucciones diferentes, dentro de las cuales se pueden incluir con facilidad todas las demás instrucciones no fundamentales. Cuanto estamos diciendo para el BASIC tiene validez para todos los lenguajes de programación (FORTRAN, COBOL, PASCAL)..., porque son aspectos intrínsecos del funcionamiento de los ordenadores digitales modernos.

Las cuatro familias de instrucciones son las siguientes:

- declarativas
- de entrada y de salida
- de cálculo y de asignación (aritméticas)
- de control

En la figura 2 puede ver una representación de estas familias y algunas de las instrucciones de BASIC asociadas a cada una.

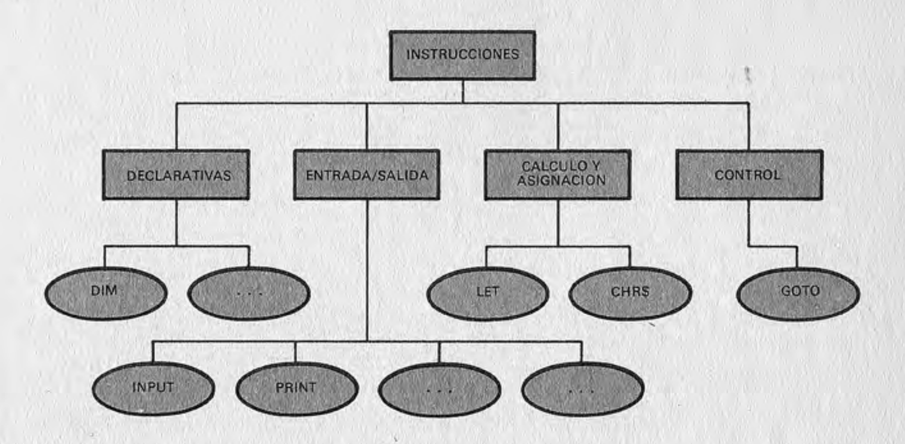


Fig. 2.—Familias de instrucciones del BASIC y algunos de sus "miembros".

Instrucciones de entrada y de salida (E/S)

Para ser prácticos, comencemos el estudio de las instrucciones del lenguaje BASIC con las que permiten escribir inmediatamente programas pequeños y sencillos: son las instrucciones de E/S (I/O, si se quiere utilizar la abreviatura de la denominación inglesa "Input/Output").

Las instrucciones declarativas que veremos inmediatamente después son muy importantes desde el punto de vista conceptual, pero son menos significativas para desarrollar los primeros ejemplos de programas.

Las instrucciones de entrada y de salida son las que permiten al ordenador recibir datos (entrada) y proporcionar resultados (salida). Si alguien piensa que son menos importantes que las de cálculo, es decir, las instrucciones que realizan los procesos propiamente dichos, se equivoca. Las instrucciones de entrada y de salida son, por muchos motivos, las más complejas en todos los lenguajes de programación. Por su empleo más o menos correcto se suele reconocer a un buen programador.

En BASIC, las instrucciones más importantes de este grupo son INPUT y PRINT. Ya vimos su utilización en algunos programas anteriores. Entre las instrucciones de entrada y de salida, si queremos ser más precisos, hay también las que ponen en comunicación el ordenador con las memorias exteriores, las grabadoras de cinta de casete o las unidades de disco flexible y también las que programan el empleo de las impresoras, de los joysticks y de los paddles (ver volumen número 1 de la B.B.I.). Todo aquello que va hacia el ordenador o que gira a su alrededor, se denomina en la jerga Entrada/Salida.

Input

La instrucción INPUT debe ir acompañada por una lista de variables. Cuando durante la ejecución del programa se llega a esta instrucción, a cada una de estas variables se le asigna el valor efectivo que introduzcamos por el teclado (vea la figura 3).

Veamos un ejemplo escribiendo un programa que calcula el producto de dos números dados por el teclado

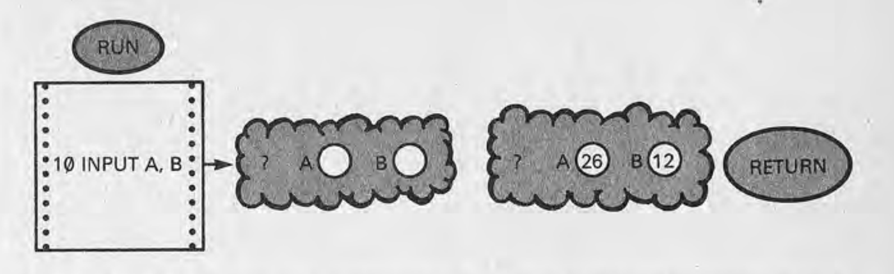


Fig. 3.—Al leer una instrucción INPUT el ordenador espera que introduzcamos los valores para cada una de las variables de su lista (y que cada valor sea del tipo adecuado a su variable).

```
10 INPUT A,B
20 C = A * B
30 PRINT C
```

En la línea 10 encontramos indicadas dos variables (A y B) que, según las reglas del BASIC son variables numéricas (repetimos que si fueran seguidas por el signo $ serían alfanuméricas o de cadena). INPUT A, B, significa: dar un valor para A y otro para B, tecleándolos durante la ejecución del programa.

La línea 20 contiene el cálculo, propiamente dicho, del producto (en los ordenadores, la multiplicación se indica casi siempre por el asterisco *). El resultado es "asignado" a la variable C, es decir, se introduce en ella.

En la línea 30, el programa hace que el valor del producto (C) se visualice en la pantalla (PRINT C).

Probemos a ejecutarlo (una vez escrito, claro).

```
RUN
?26,12
312
```

El signo de interrogación es un signo de aviso (en inglés, "prompt") que el ordenador envía a la pantalla para recordarnos que debemos responder a una instrucción de entrada. Es interesante observar que la coma que escribimos después del primer número indica simplemente una separación entre el primer dato y el segundo. Si hubiésemos querido multiplicar números fraccionarios o reales, para los que

normalmente usamos una coma (3,1416) habríamos tenido que emplear el punto en lugar de la coma como es habitual en los países anglosajones. Veamos un ejemplo en el que se quiere multiplicar 3,14 por 0,005:

```
RUN
?3.14, 0.005
0.0157
```

A este respecto debemos indicar que, excepto si se explicita lo contrario, el BASIC suele tomar toda variable numérica como de simple precisión. En caso de que no la queramos de ese tipo deberemos usar los sufijos % o # ya vistos.

La instrucción INPUT permite enviar a la pantalla un mensaje de aviso más detallado. Veamos un ejemplo:

```
10 INPUT "DAME DOS NUMEROS ",A,B
20 C=A*B
30 PRINT C
```

Es decir, el mensaje se "encierra" entre comillas.
Su ejecución es:

```
RUN
DAME DOS NUMEROS 20,3
 60
Ok
```

De este modo, sobre todo cuando en un programa hay muchas instrucciones INPUT, podemos asociar a cada una de ellas un mensaje para indicar qué es lo que nos pide y cómo deben proporcionarse los datos a la entrada.

Hemos dicho que en BASIC hay también variables que permiten procesar cadenas de caracteres. A continuación damos un ejemplo de cómo introducirlas:

```
10 INPUT "COMO TE LLAMAS? ",NOMBRE$
20 PRINT "BUENOS DIAS"
30 PRINT NOMBRE$
40 PRINT "HAS APRENDIDO EL USO DE LAS
   CADENAS"
```



```
RUN

COMO TE LLAMAS? IGNACIO
BUENOS DIAS
IGNACIO
HAS APRENDIDO EL USO DE LAS CADENAS
Ok
```

Print

Pasamos ahora a la instrucción de salida más importante: PRINT. Vimos ya en muchos ejemplos anteriores cómo se emplea. Después de la palabra PRINT se indican las variables objeto de visualización. Es posible visualizar también, de forma directa cálculos numéricos, o con cadenas, como resulta fácil comprender a partir del ejemplo siguiente:

```
10 PRINT 3*6+10
20 PRINT "HAS "+"VISTO?"
Ok
RUN
 28
HAS VISTO?
Ok
```

Cuando se visualizan variables, o datos, con la instrucción PRINT es posible "controlar" la posición en la que aparecerán en la pantalla, mediante el empleo de dos separadores: "coma" o "punto y coma", tal como indicamos con anterioridad. Veamos mejor su utilización con algunos ejemplos:

```
10 A = 234
20 B = 345
30 C = 456
40 PRINT A, B, C
RUN
234            245                 456
```

En la línea 40, las variables A, B y C están separadas por comas. Por este motivo aparecen a partir de algunas po-

siciones fijas de la pantalla (aunque diferentes para cada ordenador).

Si cambiamos las comas por puntos y comas en la línea 40, obtendremos unos números prácticamente "pegados", salvo que dejemos espacio para un eventual signo menos de los números negativos.

```
10 A = 234
20 B =345
30 C = 456
40 PRINT A; B; C
RUN
234 235 456
```

Si ponemos una coma o un punto y coma después de la última variable de una instrucción PRINT, haremos que la instrucción PRINT sucesiva no se inicie en una nueva línea, sino que continúe en la misma. Veamos un ejemplo:

```
10 A$="PACO"
20 B$="PEREZ"
30 PRINT A$;
40 PRINT B$,
50 PRINT A$,B$
RUN
PACOPEREZ       PACO            PEREZ
Ok
```

Si no hubiésemos puesto el punto y coma en la línea 30 ni la coma en la 40, hubiéramos tenido como resultado:

```
30 PRINT A$
40 PRINT B$
50 PRINT A$,B$
RUN
PACO
PEREZ
PACO            PEREZ
Ok
```

El programa de la Figura 4 es muy sencillo, pero sirve para ilustrar cómo es posible iniciarse en la realización de


```
70 REM *****************************
80 REM *   CREACION DE UN DAMERO   *
90 REM *****************************
100 L1$="**   **   **   **   **   "
110 L2$="   **   **   **   **   **"
120 PRINT L1$
130 PRINT L1$
140 PRINT L2$
150 PRINT L2$
160 PRINT L1$
170 PRINT L1$
180 PRINT L2$
190 PRINT L2$
200 PRINT L1$
210 PRINT L1$
220 PRINT L2$
230 PRINT L2$
240 PRINT L1$
250 PRINT L1$
260 PRINT L2$
270 PRINT L2$
280 PRINT L1$
290 PRINT L1$
300 PRINT L2$
310 PRINT L2$
320 END

RUN
**   **   **   **   **
**   **   **   **   **
   **   **   **   **   **
   **   **   **   **   **
**   **   **   **   **
**   **   **   **   **
   **   **   **   **   **
   **   **   **   **   **
**   **   **   **   **
**   **   **   **   **
   **   **   **   **   **
   **   **   **   **   **
**   **   **   **   **
**   **   **   **   **
   **   **   **   **   **
   **   **   **   **   **
**   **   **   **   **
**   **   **   **   **
   **   **   **   **   **
   **   **   **   **   **
Ok
```

Fig. 4.—Ejemplo de programa en BASIC con la instrucción PRINT y de su ejecución, que proporciona un damero esquematizado gracias al empleo de cadenas.

gráficos con la sola instrucción PRINT. En las líneas 100 y 110 de dicho programa se preparan dos cadenas de asteriscos. Al visualizarlas luego de forma alternada, por parejas, se obtiene en la pantalla un damero.

Las comillas que hay al comienzo y al final de los valores de variables alfanuméricas (en nuestro ejemplo antes y después de los asteriscos de L1$ y L2$) no forman parte de las cadenas propiamente dichas: son usadas por el BASIC precisamente para delimitar los caracteres que constituyen realmente la cadena.

La elección de los asteriscos se hizo exclusivamente para hacer más uniforme el damero, pero podría haberse elegido cualquier otro carácter.

Instrucciones declarativas y los arrays

Las instrucciones declarativas son de las más importantes en los lenguajes de programación modernos. Sirven para "declarar", como su propio nombre indica por anticipado, de qué tipo o características son determinadas variables y, por consiguiente, cómo podrán y deberán ser utilizadas. Gracias a estas declaraciones, todo el programa se ejecutará más correctamente y también resultará más fácil de comprender a quienes lean su listado.

En el lenguaje BASIC ordinario existe una sola instrucción de declaración: DIM, que es la abreviatura de "dimensión". Tratemos de comprender lo que significa y su forma de empleo. Para introducir la instrucción DIM debemos hablar primero de un nuevo tipo de variable que los ingleses llaman "array" (literalmente "hilera") y que podemos denominar "variable con índices" aunque se suele emplear también el término inglés, o a veces "vector", o "matriz". Como siempre, vamos a introducir el nuevo concepto a través de un ejemplo.

Supongamos que estamos en un laboratorio en donde es preciso realizar una serie de medidas experimentales. Sobre los datos de estas medidas deberemos efectuar después diversos análisis estadísticos: cálculo de los valores medio, mínimo, máximo, etc. Es, pues, necesario poder recoger los datos en una serie de variables, con las cuales

efectuar luego los cálculos. Si los datos son pocos, por ejemplo 5, podríamos escribir:

```
100 INPUT D1
110 INPUT D2
120 INPUT D3
130 INPUT D4
140 INPUT D5
```

y luego emplear las cinco variables D1, D2, D3, D4 y D5 para los cálculos. En nuestro caso, el valor medio sería:

```
200 VM = (D1 + D2 + D3 + D4 + D5) / 5
```

Los otros cálculos serían más complejos largos y tediosos, aunque factibles. Pero, ¿qué ocurriría si tuviéramos 100 ó más datos? Sería impensable utilizar otras tantas variables para contenerlos, puesto que la escritura de las operaciones se haría simplemente imposible. Para nuestra comodidad, no obstante, el BASIC nos proporciona un instrumento para resolver el problema: el array o variable con índices.

Un array no es otra cosa que un conjunto ordenado de variables simples reunidas bajo un mismo nombre. En el caso más elemental, el del array con un solo índice (o *vector*), podemos pensar en una fila de casillas numeradas, cada una de las cuales (*elemento* del array) es una variable simple (Figura 5a). En nuestro ejemplo (Figura 5b), las cinco variables diferentes D1, D2, D3, D4 y D5, se convierten en los cinco elementos D(1), D(2), D(3), D(4) Y D(5) del array D. ¿Dónde está la diferencia? Lo veremos inmediatamente con el ejemplo siguiente:

```
100 FOR I = 1 TO 5
110 PRINT D (I)
120 NEXT I
```

Este pequeño programa imprime los valores de las cinco variables. La línea 100, en la que usamos el conjunto FOR... NEXT todavía no explicado, indica que ha de repetirse todo cuanto sigue hasta la línea 120 (NEXT) hasta que el valor de I que parte inicialmente de 1 y aumenta 1 en cada paso, valga 5. La línea 100 imprime el valor del elemento número I

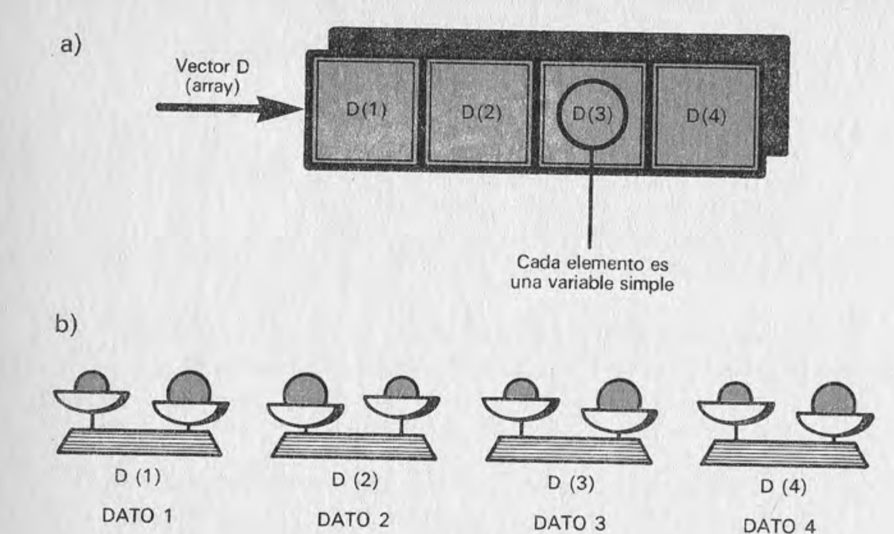


Fig. 5.—(a) Ejemplo de array de un solo índice (vector). (b) Correspondencia con los datos del laboratorio.

del array D; es decir, el valor de la I-ésima casilla del array. En la primera "vuelta", I vale 1 y por ello la línea 100 equivale a PRINT D(1), que imprime el valor elemento de D. En la segunda vuelta, I vale 2 y se imprime el valor contenido en el segundo elemento y así sucesivamente. Así, si hubiéramos medido en el laboratorio los valores 125, 259, 386, 421 y 536 la ejecución del programa será:

```
RUN
125
259
386
421
536
```

El mismo programa de tres instrucciones podría imprimir 2000 valores, con la simple sustitución de la línea 100 por FOR I = 1 TO 2000. Las instrucciones FOR y NEXT constituyen un "bucle" pero de ellas hablaremos más extensamente en este mismo capítulo al tratar de las instrucciones de control.

Los arrays no están limitados a un índice (o dimensión); se pueden tener arrays con dos índices *(matrices)*, que podemos considerar como "tableros" u hojas cuadriculadas (divididas en líneas y columnas), en las que cada casilla contiene el valor de una variable simple (figura 6) e incluso con tres o más índices. De hecho el BASIC no suele imponer límites al número de dimensiones.

De cualquier modo, los arrays más utilizados son los de una y dos dimensiones (índices), que suelen llamarse vectores y matrices, respectivamente.

Al igual que las variables, los arrays tienen también su tipo, que debe coincidir con el de los elementos que contienen y que se indica con los habituales sufijos.

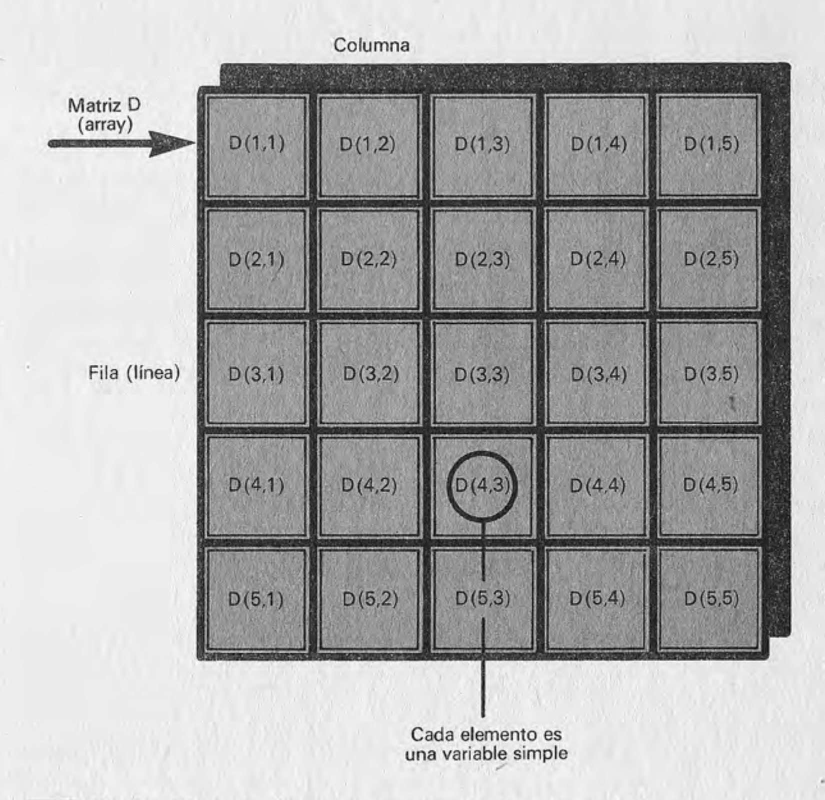


Fig. 6.—Ejemplo de array con dos índices (matriz)

Por ejemplo:

A(5)Es el quinto elemento del array numérico real A.
XY(40) Es el elemento cuadragésimo del array numérico XY.
NOMBRE$(2,3) Es el tercer elemento de la segunda fila del array alfanumérico NOMBRE$.

Los dos primeros ejemplos son de una dimensión (vectores) y el tercero tiene dos dimensiones (matriz).

Dim

Las variables con índices son, pues, variables con mayor capacidad que las normales y pueden contener muchos datos. Lo importante es que estos datos sean todos del mismo tipo: números o cadenas.

Así, si tenemos que tratar los códigos de centenares o miles de productos de un almacén no precisamos inventar centenares o miles de nombres de variables simples, sino que nos bastará un solo nombre, por ejemplo CODIGO, y avisar al ordenador de que esta variable contendrá miles de valores. Esta operación de "aviso" es precisamente una declaración de dimensionamiento y se hace en BASIC mediante la instrucción DIM. Veamos algunos ejemplos:

```
10 DIM CODIGO(1000)
10 DIM A(20,30)
10 DIM NOMBRE$(30),CALLES(30)
```

La instrucción DIM debe preceder a cualquier otra en la que aparezca la variable.

El lenguaje BASIC acepta que no dimensionemos expresamente una variable cuando su dimensión no es superior a 10. Por ejemplo, no es necesario escribir:

```
10 DIM A(6)
```

Cuando el ordenador encuentra, por primero vez en un programa, una nueva variable con un índice inferior o igual a 10, efectúa automáticamente un dimensionamiento de 10 de esta variable. Si encontrara un índice superior a 10 el BASIC dará un error por variable no dimensionada.

Veamos un ejemplo de programa que emplea la instrucción DIM, (se ilustra en la figura 7).

Supongamos que en una escuela los alumnos de una clase quieren utilizar un ordenador personal para conservar las notas obtenidas. El programa debe almacenar los nombres de los alumnos y junto a cada uno de ellos poner la nota obtenidas en la asignatura en cuestión.

Para resolver este problemas debemos emplear dos vectores: uno de cadena y otro numérico. El primero NOMBRE$, para contener los nombres y el segundo NOTAS para contener las notas. Puesto que los alumnos serán seguramente más de 10, deberemos dimensionar los dos vectores con DIM. Supongamos que los alumnos sean 30; cambiando el dimensionamiento (línea 180) se puede emplear el programa para cualquier otra clase. ¡Ojo con esto! pues, si cambiamos el número de los alumnos, deberemos hacer lo mismo en las líneas 230, 320 y 390, con las tres instrucciones de bucle FOR NEXT que veremos a continuación.

Una vez realizado el dimensionamiento de los vectores NOMBRE$ y NOTAS, el ordenador puede introducir en el primero los nombres de los alumnos (líneas del programa desde 200 a 250) y luego, después de haber solicitado la asignatura objeto de la calificación (línea 270 y 280), se cargará el vector de las notas (líneas desde 300 a 350). Finalmente, el programa imprime el "archivo" y con ello se visualizarán en la pantalla las notas.

Evidentemente un programa de esta naturaleza es algo "ingenuo", pero resulta fácil pensar que, si añadimos las instrucciones del BASIC necesarias para almacenar en disco las calificaciones y para imprimirlas luego en papel, se obtendrá un verdadero programa profesional. Estas instrucciones las veremos más adelante.

El lector imaginativo ya habrá pensado sin duda que no hay razón para limitar las calificaciones a una sola asignatura cada vez. En efecto, podríamos hacer el array NOTAS de dos dimensiones, de forma que cada línea representara un alumno y cada columna una asignatura (cuyo nombre estaría en el ahora vector ASIGNATURA$). Tendremos entonces (para, p. e., 7 asignaturas):

```
100 REM ***************************
110 REM *                         *
120 REM *   ARCHIVO DE LA CLASE   *
130 REM *                         *
140 REM ***************************
150 REM
160 REM DIMENSIONAMIENTO DE LOS
    VECTORES
170 REM     ALUMNOS Y CALIFICACIONES
180 DIM NOMBRE$(30),NOTAS(30)
190 REM
200 REM     BUCLE PARA CARGAR
210 REM LOS NOMBRES DE LOS ALUMNOS
220 PRINT "DAR NOMBRE A LOS ALUMNOS"
    :PRINT
230 FOR K=1 TO 30
240 INPUT NOMBRE$(K):REM INTRODUCCION
    DEL NOMBRE
250 NEXT K
260 REM
270 REM SOLICITUD DE LA ASIGNATURA
280 INPUT "QUE ASIGNATURA";ASIGNATURA$
290 REM
300 REM BUCLE DE CARGA DE NOTAS
310 PRINT "DAR LAS NOTAS DE ";
    ASIGNATURA$:PRINT
320 FOR K=1 TO 30
330 PRINT "NOTA DE ";NOMBRE$(K);
340 INPUT NOTAS(K)
350 NEXT K
360 REM
370 REM IMPRESION DEL REGISTRO
380 PRINT "CALIFICACIONES DE ";
    ASIGNATURA$
390 FOR K=1 TO 30
400 PRINT NOMBRE$(K),NOTAS(K)
410 NEXT K
420 REM
430 END
```

Fig. 7.—Programa "Archivo de la clase".

DIM NOMBRE$(30), NOTAS (30,7)
DIM ASIGNATURA$(7) [esta no sería imprescidible]

Por supuesto, los bucles que harían falta en esta nueva estructura (ver figura 8) no serían los mismos que, antes aunque sí parecidos. ¿Se atreve a hacer el nuevo programa? En la Figura 9 les ofrecemos una de las mucha soluciones válidas que resuelven el problema, y una ejecución. Si se fija verá como el array NOTAS se encuentra siempre "encerrado" entre dos bucles FOR... NEXT. ¿Por qué? Lo que hacemos es que el primer bucle (p.e. el que comienza en la línea 280) controla la variación del alumno (primer índice de NOTAS, como se ve en la Figura 8) y el segundo bucle (p.e. en la línea 300) hace lo propio con las asignaturas. De esta forma, para cada uno de los alumnos (FOR K = 1 to NAL) tomamos todas sus notas [FORL=1 TO NAS] antes de pasar al siguiente. Esta estructura, denominada de "bucles anidados", la veremos más en detalle al tratar de la instrucción FOR... NEXT; será sin duda un buen momento para repasar de nuevo este programa

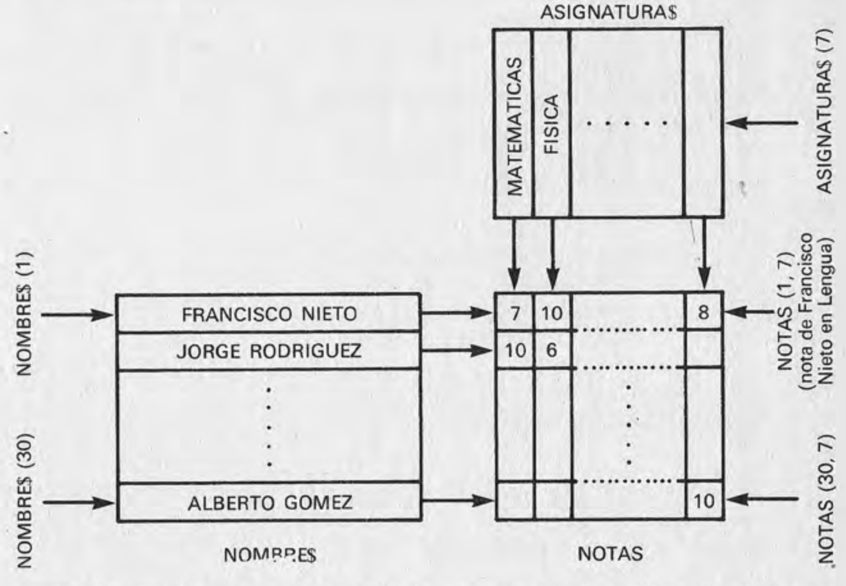


Fig. 8.—Nueva estructura de los arrays usados al "ampliar" el programa de la Figura 7.

```
10 REM **************************
20 REM *  ALUMNOS Y ASIGNATURAS  *
30 REM **************************
40 INPUT "NUMERO DE ALUMNOS";NAL
50 INPUT "NUMERO DE ASIGNATURAS";NAS
60 PRINT
70 REM **************************
80 REM *  DIMENSIONO LOS ARRAYS  *
90 REM **************************
100 DIM NOMBRE$(NAL),NOTAS(NAL,NAS),ASIGNATURA$(NAS)
110 REM **************************
120 REM *  NOMBRE DE LOS ALUMNOS  *
130 REM **************************
140 FOR I=1 TO NAL
150 PRINT "NOMBRE DEL ALUMNO NUMERO";I;
160 INPUT NOMBRE$(I)
170 NEXT I
180 REM  ******************************
190 REM * NOMBRES DE LAS ASIGNATURAS  *
200 REM *******************************
210 FOR J=1 TO NAS
220 PRINT "NOMBRE DE LA ASIGNATURA ";J;
230 INPUT ASIGNATURA$(J)
240 NEXT J
250 REM ******************************************
260 REM *  NOTAS PARA CADA ALUMNO Y ASIGNATURA   *
270 REM ******************************************
280 FOR K=1 TO NAL
290 PRINT "ALUMNO D.";NOMBRE$(K)
300 FOR L=1 TO NAS
310 PRINT "NOTA EN ";ASIGNATURA$(L);
320 INPUT NOTAS(K,L)
330 NEXT L
340 NEXT K
350 REM ***********************
360 REM *  SALIDA DEL ARCHIVO  *
370 REM ***********************
380 PRINT "NOTAS OBTENIDAS EN:"
390 PRINT,
400 FOR I=1 TO NAS
410 PRINT ASIGNATURA$(I),
420 NEXT I
430 PRINT
440 PRINT "POR LOS ALUMNOS SIGUIENTES:"
450 FOR J=1 TO NAL
460 PRINT "D. ";NOMBRE$(J),
470 FOR K=1 TO NAS
480 PRINT NOTAS(J,K),
490 NEXT K
```



```
500 PRINT
510 NEXT J
520 END
```

```
NUMERO DE ALUMNOS? 5
NUMERO DE ASIGNATURAS? 4

NOMBRE DEL ALUMNO NUMERO 1 ? F.RUIZ
NOMBRE DEL ALUMNO NUMERO 2 ? E.ZAMORA
NOMBRE DEL ALUMNO NUMERO 3 ? A.BAUTISTA
NOMBRE DEL ALUMNO NUMERO 4 ? F.NIETO
NOMBRE DEL ALUMNO NUMERO 5 ? C.GONZALEZ
NOMBRE DE LA ASIGNATURA  1 ? FISICA
NOMBRE DE LA ASIGNATURA  2 ? QUIMICA
NOMBRE DE LA ASIGNATURA  3 ? ALGEBRA
NOMBRE DE LA ASIGNATURA  4 ? GEOLOGIA

ALUMNO D.F.RUIZ
NOTA EN FISICA? 5
NOTA EN QUIMICA? 5
NOTA EN ALGEBRA? 6
NOTA EN GEOLOGIA? 3
ALUMNO D.E.ZAMORA
NOTA EN FISICA? 4
NOTA EN QUIMICA? 5
NOTA EN ALGEBRA? 5
NOTA EN GEOLOGIA? 4

ALUMNO D.A.BAUTISTA
NOTA EN FISICA? 8
NOTA EN QUIMICA? 7
NOTA EN ALGEBRA? 9
NOTA EN GEOLOGIA? 5
ALUMNO D.F.NIETO
NOTA EN FISICA? 5
NOTA EN QUIMICA? 6
NOTA EN ALGEBRA? 8
NOTA EN GEOLOGIA? 5
ALUMNO D.C.GONZALEZ
NOTA EN FISICA? 3
NOTA EN QUIMICA? 5
NOTA EN ALGEBRA? 4
NOTA EN GEOLOGIA? 10
```

```
NOTAS OBTENIDAS EN:
              FISICA        QUIMICA       ALGEBRA       GEOLOGIA

POR LOS ALUMNOS SIGUIENTES:
D. F.RUIZ      5             5             6             3

D. E.ZAMORA    4             5             5             4

D. A.BAUTISTA  8             7             9             5

D. F.NIETO     5             6             8             5

D. C.GONZALEZ  3             5             4             10

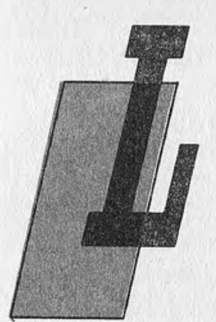
Ok
```


Fig. 9.—Una de las posibles soluciones, y su ejecución, para ampliar el programa de la Figura 7 a 7 asignaturas.

CAPITULO IV

INSTRUCCIONES ARITMETICAS Y MANEJO DE CARACTERES

Instrucciones de asignación


Las instrucciones de asignación son las más sencillas, pero permiten desarrollar el trabajo efectivo que pedimos a un ordenador. Con estas instrucciones se pueden realizar cálculos matemáticos con datos numéricos y manejar las cadenas de caracteres.

La instrucción más típica de este grupo es la instrucción LET; que permite atribuir un valor a una variable. Vimos ya algunos ejemplos de asignación, pero consideramos de utilidad ver algunos otros:

```
10 LET A=35
20 LET B=12
30 LET F$="JOSE"
40 PRINT A,B,F$
Ok
RUN
 35             12             JOSE
Ok
```

La palabra LET (que significa "poner" o "hacer") fue utilizada en las primeras versiones del BASIC, pero actualmente en la mayor parte de los ordenadores no es necesario uti-

lizarla. Basta escribir directamente el nombre de una variable a la izquierda del signo igual.

```
10 A=(10+25)*3
20 B=38-3
30 C=A-B
40 R$="RESULTADO="
50 PRINT R$;C
Ok

RUN
RESULTADO= 70
Ok
```

El ZX Spectrum es uno de los pocos ordenadores donde esto no cabe. Al tener un teclado especial, cuyas teclas llevan grabadas las palabras reservadas de su dialecto BASIC, requiere el empleo obligatorio de la instrucción LET.

La existencia del signo igual (=) en las instrucciones LET nos obliga a hacer inmediatamente una puntualización importante. El signo igual no representa en absoluto una igualdad matemática, en el sentido que aprendimos en la escuela.

El significado de este signo en los ordenadores se explica como sigue: calcular primero el valor de todo lo que está a la derecha del signo igual según la expresión correspondiente y luego asignar el resultado a la variable que está a la izquierda. En el primer miembro deberá estar siempre una sola variable y por ello, escribir en BASIC:

```
10 56 * 3 = B
10 A = 10 = R
```

es absolutamente erróneo. Sin embargo podemos hacer una cosa en principio tan "rara" como:

```
10 A = A + 10
```

¿Cómo interpreta esto el ordenador? Primero calcula lo que hay a la derecha del igual, es decir, suma al valor de A,10. Luego ese resultado es asignado a la variable A borrando su anterior valor. En este caso, por ejemplo, habremos incrementado en 10 unidades el valor de A.

Otras instrucciones de asignación del BASIC (DATA, READ y RESTORE) las dejaremos para el capítulo siguiente.

Instrucciones de cálculo

Hemos dicho que a la derecha del signo igual pueden escribirse expresiones numéricas o alfanuméricas. Vamos a ver cuáles son las reglas para escribirlas en BASIC.

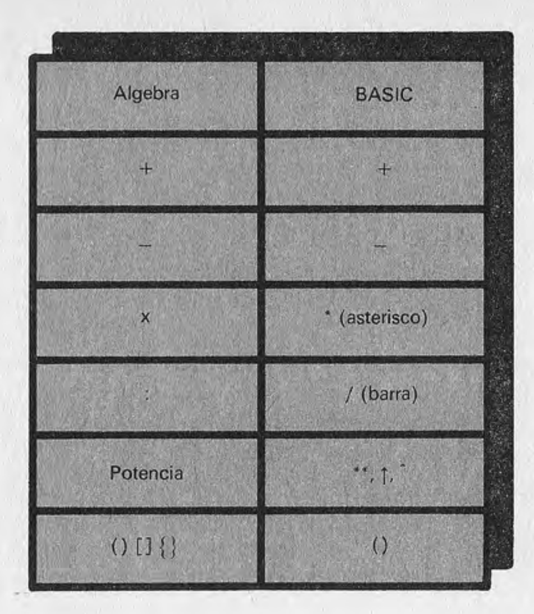
Las *expresiones matemáticas*, en BASIC, son prácticamente idénticas a las que se estudian en la escuela, con algunas pequeñas diferencias en la escritura de los operadores (los símbolos de las operaciones).

Tomemos el ejemplo de una expresión algebraica, escrita como hacemos normalmente y probemos a volverla a escribir el el lenguaje BASIC. Para ello, tenemos que emplear los símbolos de las operaciones algebraicas (*operadores algebráicos*) admitidos por el BASIC (Figura 1).

A + B	A + B
A − B	A− B
A x B A · B A B	A*B
$\frac{A}{B}$ A:B	A/B
A^B	A**B A↑B A^B

Como se puede constatar, las operaciones de *suma* y *resta* se indican como es habitual con "+" y "−".

El producto se indica, en cambio, por el asterisco "*" y no por "x" o el punto como se emplea en matemáticas. A menudo también indicamos el *producto* de dos valores simplemente escribiéndolos uno después de otro. Así, con "a b" queremos decir "a por b". Esto no vale en BASIC. Nunca podemos dejar algo sobreentendido en informática, por lo que



Algebra	BASIC
+	+
–	–
x	* (asterisco)
:	/ (barra)
Potencia	**, ↑, ^
() [] {}	()



Fig. 1.—Operadores algebráicos

el producto debe indicarse de forma expresa mediante el asterisco: A * B (obsérvese que en BASIC se utilizan las letras mayúsculas).

La *división* viene indicada por el carácter "/" (en inglés, slash). Los dos puntos ":" tienen en BASIC un significado completamente distinto que no coincide en todas las versiones.

La *elevación a una potencia* es otra operación con un símbolo particular. No se escribe el exponente como un número más pequeño colocado arriba a la derecha de la base como "4^3". Se sitúan la base y el exponente en la misma línea, separados por uno de los símbolos "**", "↑" o "^", es decir, para nuestro ejemplo, 4 **3, 4↑3 ó también 4^3.

En lo que respecta a los paréntesis, los únicos que se utilizan en las expresiones son los redondos, que desempeñan la misma función que en matemáticas.

En la Figura 2 hemos indicado las *funciones matemáticas* más comunes. Observe que la raíz cuadrada de un número X en BASIC se indica por SQR(X). "SQR" es la abreviatura de las palabras inglesas "square root", que quiere decir "raíz cuadrada".

Matemáticas	BASIC
e^x	EXP(X)
$\log_e x$	LOG(X)
$\sqrt{x}$	SQR(X)
sen(x)	SEN(X)
cos(x)	COS(X)
tg(x)	TAN(X)

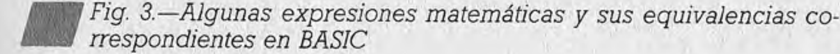
Fig. 2.—Funciones matemáticas

Otras funciones matemáticas que suelen encontrarse inmediatamente en BASIC son la exponencial EXP(X) y su inversa LOG(X), que es el logaritmo en base "e" (cuidado con esto, pues en la notación de "andar por casa" usábamos LOG como logaritmo en base 10 y LN como el de base "e"), junto a las funciones trigonométricas seno, conseno y tangente, SIN(X), COS(X) Y TAN(X) respectivamente. De estas últimas funciones es preciso recordar que normalmente se refieren a ángulos expresados en radianes (2 π radianes = 360 °). Si tenemos un ángulo expresado en grados, para calcular su valor en radianes basta multiplicarlo por:

Xrad = Xgrad* (3.1416/180).

En la figura 3 damos algunos ejemplos de expresiones matemáticas escritas en BASIC.

Matemáticas	BASIC
$\frac{x+y}{x+z}$	(X + Y) / (Y + Z)
$3 + [(5 + A)(B + C)]$	3 + ((5 + A) · (B + C))
$\log_e 528$	LOG (528)
$\sqrt{\frac{1+x}{1+y}}(\text{tg}(y) + e^{x-2})$	SQR((1 + X) / (1 + Y)) * (TAN(Y) + + EXP (X − 2))


Fig. 3.—Algunas expresiones matemáticas y sus equivalencias correspondientes en BASIC

Ejemplos de programas matemáticos

Veamos ahora un programa en el que se calculan algunas expresiones algebraicas. La primera sirve para obtener el volumen de un paralelepípedo y la segunda para el de una esfera. El listado correspondiente es como sigue:

```
10 PRINT "PROGRAMA PARA EL CALCULO DE VOLUMENES"
20 PRINT "------------------------------------"
30 PRINT
40 PRINT "CALCULO DEL VOLUMEN DE UN PARALELEPIPEDO"
50 INPUT "ALTURA";H
60 INPUT "LONGITUD";LG
70 INPUT "ANCHURA";LA
80 V=LA*LG*H
90 PRINT "EL VOLUMEN ES ";
100 PRINT V
110 PRINT
120 PRINT "CALCULO DEL VOLUMEN DE UNA ESFERA"
130 INPUT "RADIO DE LA ESFERA";R
```



```
140 V=3.1416*(4/3)*R^3
150 PRINT "EL VOLUMEN DE LA ESFERA ES ";
160 PRINT V
170 END
```

El programa pide las dimensiones del paralelepípedo (altura, longitud y anchura) y luego, en la línea 80 las multiplica entre sí para calcular el volumen y asigna el resultado a la variable V, sacando a pantalla el resultado. A continuación, pide el radio de una esfera y calcula su volumen en la línea 140.

Observe que se emplea el mismo nombre de variable (V) tanto para el volumen del paralelepípedo como para el de la esfera. Esto es posible con tal de que se utilice de forma correcta, es decir, que sepamos que no interfieren ambos usos.

Tratemos de aclarar este punto. Inicialmente, V no contiene nada. En la línea 80 se le asigna el valor del volumen del paralelepípedo. Inmediatamente después, en la línea 100, se visualiza con PRINT V su contenido. Posteriormente (línea 140) se calcula el volumen de la esfera que se asigna a la misma variable V; desde este preciso momento, V contiene solamente el volumen de la esfera y el del paralelepípedo ha sido borrado. En efecto, ahora solo nos interesa el volumen de la esfera que se visualizará con PRINT V (línea 160). Si, por ejemplo, hubiéramos deseado visualizar los valores de los dos volúmenes al tiempo hubiera sido necesario asignar el valor del de la esfera a otra variable que no fuera V (p.e. VE) para poder conservar el valor del volumen del paralelepípedo y poder hacer un PRINT V, Ve.

Veamos un ejemplo de ejecución del programa:

```
RUN
PROGRAMA PARA EL CALCULO DE VOLUMENES
-------------------------------------

CALCULO DEL VOLUMEN DE UN PARALELEPIPEDO
ALTURA? 5
LONGITUD? 6
ANCHURA? 8
EL VOLUMEN ES  240

CALCULO DEL VOLUMEN DE UNA ESFERA
```

```
RADIO DE LA ESFERA? 5
EL VOLUMEN DE LA ESFERA ES  523.6
Ok
```

Este ejemplo confirma que las variables en Informática se comportan como cajas en las que se pueden meter cosas "adecuadas" cuando queramos, pero teniendo en cuenta que el último objeto introducido "echa a la calle" al que estaba antes.

Veamos un segundo programa, ejemplo de cómo se puede llevar una pequeña contabilidad y calcular valores porcentuales.

Supongamos que hemos hecho un viaje y queremos saber cómo se han repartido los gastos, en porcentajes, entre el automóvil, el hotel y los restaurantes. Después de haber recibido los datos por el teclado, en la línea 90 se calcula el total de los gastos, que se conserva en la variable T. En las líneas 100, 110 y 120 se calculan los tres porcentajes y se asignan, respectivamente a las variables APERC, HPERC y RPERC.

```
10 REM PROGRAMA PARA EL CALCULO
20 REM DE LA DISTRIBUCION PORCENTUAL
30 REM DE GASTOS
40 REM CON OCASION DE UN VIAJE
50 REM
60 INPUT "GASTOS AUTOMOVIL ";A
70 INPUT "GASTOS HOTEL ";H
80 INPUT "GASTOS RESTAURANTE ";R
90 T=A+H+R
100 APORC=A*(100/T)
110 HPORC=H*(100/T)
120 RPORC=R*(100/T)
130 REM
140 PRINT "GASTOS AUTOMOVIL EN %=";
    APORC
150 PRINT "GASTOS HOTEL EN %=";HPORC
160 PRINT "GASTOS RESTAURANTE EN %=";
    RPORC
170 END
```


prueba de ejecución:

```
GASTOS AUTOMOVIL ? 20000
GASTOS HOTEL ? 120000
GASTOS RESTAURANTE ? 35000
GASTOS AUTOMOVIL EN %= 11.42857
GASTOS HOTEL EN %= 68.57143
GASTOS RESTAURANTE EN %= 20
Ok
```

Este último ejemplo quizá les pareciera banal y tengan ganas de preguntarnos por qué utilizar un ordenador, o escribir un programa, para cálculos que se realizan con mayor facilidad mediante una calculadora de bolsillo. La respuesta es que se trata solamente de unos ejemplos, que tenían que ser lo más sencillos posible. Lo que cuenta, por ahora, es el principio. Un programa efectivo podría realizarse, más o menos, del mismo modo; trabajar no con tres elementos, sino con un centenar, recibir los datos no a través del teclado, sino a partir de un fichero de disco, e incluso podría trazar también un "gráfico" de la distribución de los gastos o *extrapolar* (predecir sobre la base de los datos anteriores) previsiones para los años sucesivos, pero el sistema de programación aplicado sería el mismo.

Variables alfanuméricas. Funciones ASC y CHR$

Una variable numérica contiene números y una variable alfanumérica o de cadena, contiene caracteres. Estos caracteres, para ser almacenados en el ordenador (que, recordemos, es una máquina que sólo maneja información en forma digital) deben estar codificados en una forma numérica apropiada. A cada carácter se le asocia un número según el código particular elegido.

Entre los numerosos códigos posibles, los ordenadores personales han adoptado *el código ASCII* (ver el Apéndice A y repasar cuanto se dijo en el primer volumen de nuestra colección), código que hace corresponder a las letras, cifras, signos de puntuación y demás caracteres números que van desde el 0 al 255. Por ejemplo, las 26 letras del alfabeto inglés están codificadas con los números 65 a 91, mientras

que las diez cifras decimales están codificadas desde el 48 al 57 y la coma con el valor 44. Gracias a este código los ordenadores pueden contener y procesar caracteres, palabras y literales enteros.

No obstante, para escribir programas que traten cadenas de caracteres, no es necesario que nos preocupemos de codificar estas en ASCII, porque dicha operación se realiza de forma automática por el ordenador. Basta pulsar en el teclado las teclas correspondientes a los caracteres deseador para que el ordenados los transforme y almacene bajo la forma del código ASCII. Lo anterior es válido para los caracteres existentes en el teclado. Como se verá, también puede hacerse algo parecido con los demás.

ASC y CHR$

Con este propósito vamos a hablar inmediatamente de dos importantes funciones del lenguaje BASIC que transforman un carácter en su código numérico ASCII y viceversa. Dichas funciones son ASC y CHR$. Algunos ordenadores llaman a la primera CODE; en todo caso desempeña la misma función, que es la de proporcionar el código ASCII de un carácter. Por ejemplo, si escribimos:

```
10 PRINT ASC(A)
```

obtendremos:

```
RUN
65
```

que es precisamente el código de la letra A.

Por el contrario, la función CHR$ proporciona un carácter a partir de su código ASCII. Probemos con un ejemplo:

```
10 PRINT CHR$(66)
RUN
B
```

La función CHR$ es de gran importancia en BASIC porque permite "construir" los caracteres que no podemos introducir por el teclado para su inserción en un programa.

El más importante de todos estos caracteres es el de Return, que hace terminar la introducción de una línea o de un dato. Si tratamos de utilizarlo como cualquier otro e introducirlo entre comillas (como se hace con una cadena normal, es decir, poner comillas y pulsar la tecla de RETURN), no lo conseguiremos, porque apenas lo hayamos pulsado desempeñará el cometido de introducir un "retorno de carro" en el ordenador. Si consultamos la tabla del código ASCII encontramos que el Return tiene el código 13. Así, para introducir un Return en cualquier programa basta escribir en BASIC CHR$(13).

Para familiarizarnos con las funciones ASC y CHR$ veamos ahora dos programas: Secretísimo 1 y Secretísimo 2. El primero, como los mejores agentes de espionaje, traduce una frase a un "código secreto" (con ASC), mientras que el segundo la reconvierte a lenguaje normal (con CHR$).

He aquí el programa Secretísimo 1:

```
5 REM  *************************
10 REM *     SECRETISIMO  1     *
15 REM *************************
20 PRINT "DAME LA CLAVE SECRETA (1-37)";
30 INPUT K:IF (K<1) OR (K>37) THEN GOTO
   20:REM CONTROL CLAVE
40 PRINT "ESCRIBE UNA LETRA A CIFRAR
   (A-Z,#PARA ACABAR)";
50 INPUT A$:IF(A$<"A") OR (A$>"Z") THEN
   GOTO 40:REM CONTROL
60 IF A$="#" THEN END
70 B=ASC(A$)+K
80 PRINT B
90 GOTO 50
```

Comentemos algunas de sus particularidades:

Las líneas 20-30 piden la "clave secreta", que se asignara a la variable K.

En la línea 30, en particular, se puede observar cómo se construye una *entrada controlada*, que admite solamente los valores que entran en el margen señalado (si la clave no está entre 1 y 37, se repite la pregunta de la línea 20).

Las líneas 40-50, en donde el programa pide una letra a cifrar, son otro ejemplo de entrada controlada. No obstante,

el control es diferente al anterior, puesto que compara cadenas en lugar de números. Una expresión lógica del tipo IF A$<"A" (IF significa en inglés el si condicional castellano) es verdadera si el primer operando (el contenido de A$) precede en orden alfabético (en realidad en el orden de los códigos ASCII, pero es lo mismo), al segundo operando ("A"). En los ordenadores que no admiten este tipo de operación, es preciso comparar los códigos ASCII; en este caso, la entrada se escribiría de esta otra forma:

```
50 INPUT A$:IF (ASC(A$)<ASC("A")) OR
   (ASC(A$)>ASC("Z")) THEN GOTO 40
```

La línea 60 permite terminar el trabajo de cifrado, puesto que cuando se escribe "#" se ejecutará la instrucción END y terminará el programa. Si el ordenador no admite una instrucción END en medio del programa, es preciso escribir

```
60 IF A$="#" THEN GOTO 100
100 END
```

En la línea 70 por último, la letra se cifra sumando K al valor de su código ASCII. La línea 80 imprime el "cifrado secreto" que ha de transmitirse al "comando operativo". Por ejemplo, si K = 4, se obtendrá una conversión de este tipo:

A	E	R	E	0
69	73	86	73	79

Se observa que cada valor numérico se obtiene sumando al código ASCII la clave K = 4 (A = 65 + 4 = 69).

Con el programa Secretísimo 2, el mensaje secreto se descifra. Solamente el "comando operativo" que conoce la clave, puede efectuar la decodificación.

Y he aquí el programa Secretísimo 2:

```
5 REM **************************
10 REM *    SECRETISIMO 2       *
15 REM **************************
20 PRINT "DAME LA CLAVE SECRETA (1-37)";
30 INPUT K:IF (K<1) OR (K>37) THEN GOTO
   20:REM CONTROL CLAVE
```



```
40 PRINT "ESCRIBE UN CODIGO A DESCIFRAR
   (0,PARA ACABAR)";
50 INPUT C
60 IF C=0 THEN END
70 B$=CHR$(C-K)
80 PRINT B$
90 GOTO 50
```

En las líneas 20-30 se solicita, como es habitual, la clave secreta.

Las líneas 40-50 piden el código a descifrar. La línea 60 termina el trabajo si se ha pulsado 0.

La línea 70 reconstruye el carácter original realizando una especie de "decodificación"; primero se resta K (las operaciones entre paréntesis tienen siempre la prioridad) y luego, se asigna a B$ el carácter que tiene como código ASCII el valor calculado, que no es otro si no el carácter originalmente cifrado con el programa Secretísimo 1. Por tanto las correspondencias serán, entre otras, para K =4:

69	73	86	73	79
A	E	R	E	0

Operaciones con variables alfanuméricas. Instrucciones +, LEFT$, RIGHT$, MID$ y LEN

Existe un cierto parentesco entre variable alfanumérica y array: la primera es una secuencia ordenada de caracteres (figura 4), mientras que la segunda está constituida por una serie de elementos numéricos.

Dos cadenas pueden ser *concatenadas*, es decir, pueden ser "sumadas" para obtener una cadena resultante cons-

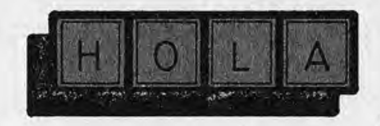


Fig. 4.—Una cadena, a semejanza de un array, contiene una secuencia de elementos, cada uno de los cuales es un carácter.

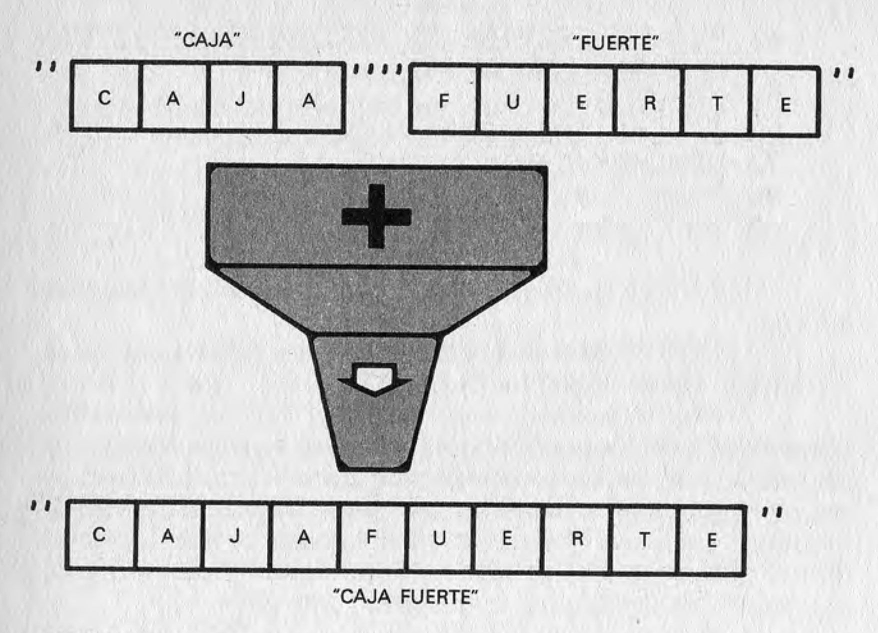


Fig. 5.—Dos cadenas pueden concatenarse para formar una más larga.

tituída por las dos originales, colocadas una tras otra, tal como se ilustra en la Figura 5.

El operador de concatenación de las cadenas suele ser el "+" pero también puede ser otro carácter, por ejemplo "&", dependiendo del ordenador que se emplee. Excepto el Sinclair QL que usa el "&", la mayor parte de los ordenadores personales que pueden realizar la concatenación emplean el "+". Veamos el sencillo prógrama correspondiente a la Figura 5:

```
100 A$="CAJA"
110 B$="FUERTE"
120 C$=A$+B$:REM CONCATENACION DE
    CADENAS
130 PRINT C$

RUN
CAJAFUERTE
OK
```


También es posible formar una cadena a partir de los caractres "extraídos" de otra, por medio de las funciones LEFT$, RIGHT$ Y MID$ (los nombres de las funciones terminan con "$", por cuanto restituyen cadenas).

LEFT$ restituye una cadena constituida por tantos caracteres como indiquemos de la parte izquierda (en inglés "left") de la cadena indicada, tal como ilustra la Figura 6. Veamos un ejemplo:

```
100 A$="JOSE LUIS"
110 B$=LEFT$(A$,4):REM EXTRAE 4
    CARACTERES
120 PRINT B$

RUN
JOSE
Ok
```

RIGHT$ hace exactamente lo mismo, pero por la parte derecha ("right", en inglés) como se ve en la Figura 7, que equivale a:

```
100 A$="JOSE LUIS"
110 B$=RIGHT$(A$,4)
120 PRINT B$

RUN
LUIS
Ok
```

La función MID$ extrae un número especificado de caracteres, a partir de la posición dada, como muestra el siguiente ejemplo (correspondiente a la Figura 8):

```
100 A$="JOSE LUIS"
110 B$=MID$(A$,2,7):REM EXTRAE 7
    CARACTERES APARTIR DEL 2
120 PRINT B$

RUN
OSE LUI
Ok
```

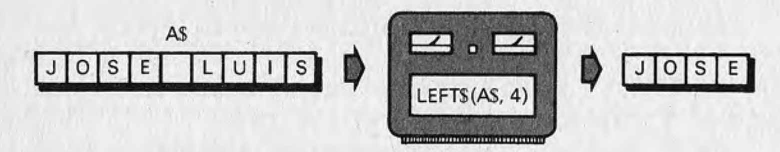


Fig. 6.—La función LEFT$ extrae caracteres de la parte izquierda de la cadena.

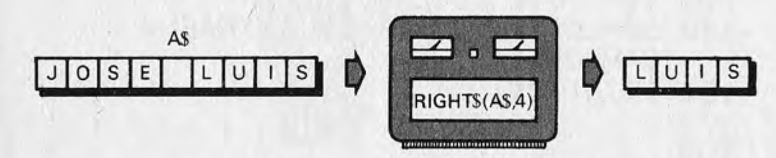


Fig. 7.—RIGHT$ extrae caracteres por la derecha de la cadena.

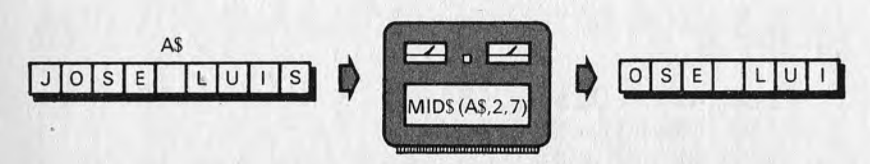


Fig. 8.—MID$ toma los caracteres que le digamos a partir de uno dado.

Como podemos ver, el primer número especifica el carácter desde el que se parte, inclusive y el segundo es el número de caracteres a extraer. El primer carácter de una cadena es el número uno.

Escribamos ahora un programa que "vuelque" una cadena, cambiando el orden de sus caracteres. Podemos hacerlo extrayendo los caracteres, uno a uno, desde el final de la cadena inicial y añadiéndolos sucesivamente a la nueva cadena. Para hacerlo, no obstante, tenemos que saber cuántos caracteres han de extraerse, es decir, la longitud de la cadena.

La longitud de una cadena se puede conocer con la fun-

ción LEN, que proporciona el número de caracteres contenidos en la cadena. Por ejemplo:

```
PRINT LEN ("ABELARDO")
 8
```

(Recordemos que cuando no ponemos número de línea la sentencia se ejecuta directamente).

De esta forma el listado sería:

```
100 A$="ROMA"
110 L=LEN(A$)
120 FOR I=L TO 1 STEP -1
130 B$=B$+MID$(A$,I,1)
140 NEXT I
150 PRINT B$

RUN
AMOR
Ok
```

Dejamos como ejercicio al lector la comprensión de este pequeño programa, aunque forzosamente hemos de remitirle al estudio del bucle FOR/NEXT del que hablaremos en el Capítulo siguiente.

Si bien estas instrucciones (LEFT$, RIGHT$ y MID$) están prácticamente normalizadas, el sinclair QL y el ZX-Spectrum las sustituyen por una propia: la función TO (en inglés "a") que toma todos los caracteres comprendidos entre los dos que le demos (inclusives). Por ejemplo (figura 9).

```
100 A$="JOSE LUIS"
110 B$=A$(2 TO 8)
120 PRINT B$
```

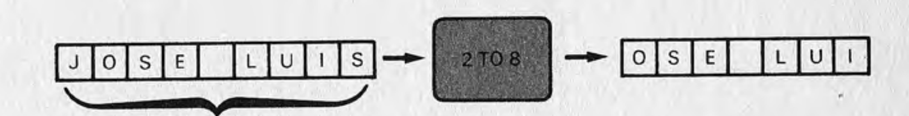


Fig. 9.—Forma en la que trabajan el QL y el ZX-Spectrum con las cadenas.

```
RUN
OSE LUI
OK
```

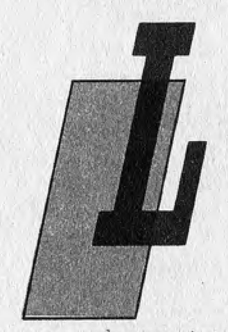
produce el mismo efecto que vimos tenía MID$ (A$,2,7), en este caso. ¿Cómo consigue el equivalente a LEFT$ y RIGHT$? Pues mencionando sólo uno de los dos parámetros. Así:

A$ (TO 4) equivaldría al LEFT$ (A$,4) que usamos antes.
A$ (6 TO) sería como el RIGHT$ (A$,4) del ejemplo anterior.

CAPITULO V

ULTIMAS INSTRUCCIONES DE ASIGNACION Y ALGUNAS DE CONTROL (SALTOS Y BUCLES)

Las instrucciones DATA, READ y RESTORE



Las instrucciones DATA, READ y RESTORE permiten introducir en el programa listas de datos que se leerán de modo análogo a como se hacía con una instrucción INPUT, que los recibía desde el exterior.

Estas instrucciones, aunque se comporten como las de entrada, son, en realidad, intrucciones de asignación por cuanto que operan solamente en el interior de la memoria del ordenador.

A menudo resulta necesario *inicializar* las variables; es decir, asignarles valores iniciales definidos. Esto se puede conseguir con facilidad mediante secuencias de instrucciones del tipo:

```
10 A = 230
20 B = 40
30 C = 155
40 D = 12
50 E = 130
60 F = 18
70 G = 2
80 H = 760
```

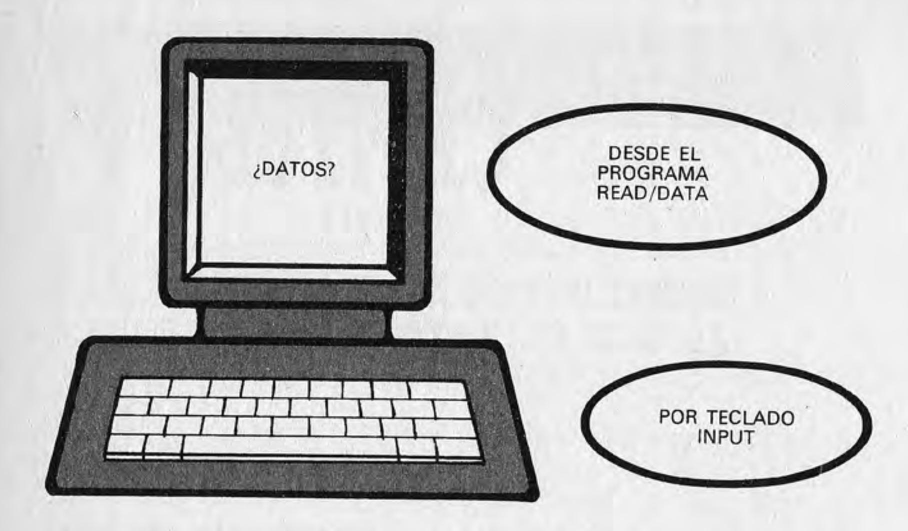


Fig. 1.—Cuando el ordenador necesita datos puede tomarlos bien desde el teclado (INPUT) o del propio programa READ/DATA

La instrucción DATA permite, por el contrario, reunir todos los datos en un solo punto del programa (normalmente al principio o al final), de forma que son más legibles:

```
10 DATA 230, 40, 155, 12, 130, 18, 2, 760
```

La instrucción READ (lee) es equivalente a la INPUT, con la diferencia de que los datos no se leen desde el teclado, sino en las líneas de la instrucción DATA. Por consiguiente se puede escribir:

```
120 READ A, B, C, D, E, F, G, H
```

y los ocho valores anteriores se asignarán a las variables correspondientes.

Veamos un sencillo ejemplo de su utilización: una agenda telefónica. Puesto que todavía no hemos hablado de cómo grabar y releer datos a partir de una cinta o de un disco, podemos tener los datos en el propio programa.


```
5 REM  *************************
10 REM *   AGENDA TELEFONICA    *
20 REM *************************
30 INPUT "NOMBRE";NOMBRE$:IF NOMBRE$="
   " THEN END
40 RESTORE
50 READ A$,TE$
60 IF A$="FINAL DE AGENDA" THEN PRINT "
   NO ESTA SU TELEFONO":GOTO 30
70 IF A$<>NOMBRE$ THEN GOTO 50
80 PRINT "EL TELEFONO DE ";NOMBRE$;" ES
   : ";TE$:GOTO 30

1000 REM *************
1001 REM *   AGENDA   *
1002 REM *************
1010 DATA "INGELEK","2505820"
1020 DATA "CASA","2022122"
1030 DATA "BANCO","7383829"
1040 DATA "INFORMACION","003"
9000 REM ********************
9001 REM *  FINAL DE AGENDA  *
9002 REM ********************
9010 DATA "FINAL DE AGENDA"," "

RUN
NOMBRE? INGELEK
EL TELEFONO DE INGELEK ES: 2505820
NOMBRE? JOSE LUIS
NO ESTA SU TELEFONO
NOMBRE?
OK
```

En primer lugar, podemos observar que los nombres y los números de teléfono se introducen en forma de cadenas contenidas en líneas de DATA (se puede continuar a voluntad introduciendo nombres y teléfonos después de la línea 1040) que se guardan junto con el propio programa (forman parte de él en realidad).

Al dar RUN el programa se inicia pidiendo (línea 30) el nombre de la persona cuyo número telefónico queremos conocer, asignándolo a la variable NO$. En este punto, la ca-

dena NO$ se compara con la cadena nula (""): si nos limitamos a teclear CR, el programa ejecutará una instrucción END y terminará.

La línea 40 ejecuta una instrucción RESTORE, que lleva al comienzo de los DATA, la lectura de las instrucciones, READ. En nuestro caso, lleva a la línea 1010, en donde se encuentra la primera instrucción DATA. La instrucción RESTORE es necesaria, porque la agenda se lee desde el principio.

La línea 50 lee dos cadenas, situadas en las dos primeras sentencias DATA que halle "INGELEK" y "2505820" respectivamente) asignándolas a las variables A$ y TE$. Después de esta operación, el *puntero* de las instrucciones DATA se habrá desplazado; es decir, la siguiente instrucción READ leerá a partir de la primera constante DATA todavía no usada.

La línea 60 verifica si, a fuerza de leer instrucciones DATA, hemos llegado a la última ("FINAL AGENDA", línea 9010).

En este caso, advierte que el nombre no está en la ageda y vuelve al comienzo (línea 30). La línea 9010 contiene también una cadena nula (""), para evitar errores en la lectura de la última variable TE$ en la línea 50.

La línea 70 compara el nombre buscado con el leído en DATA; si no coinciden, va a la línea 50 en donde se lee el siguiente.

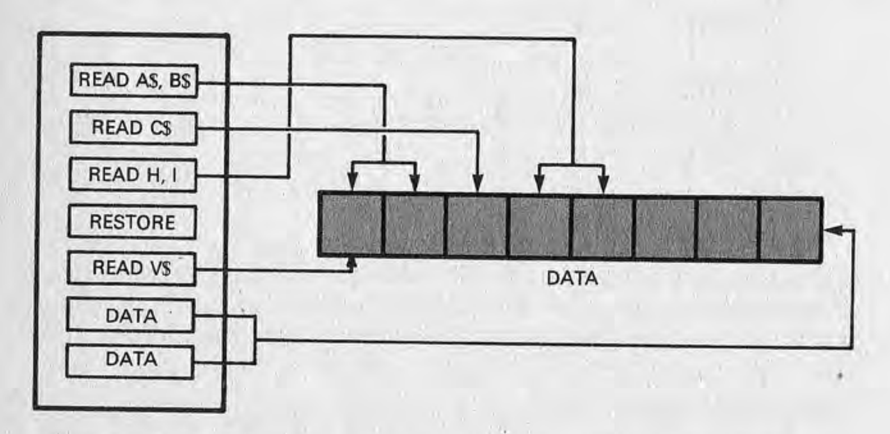


Fig. 2.—Funcionamiento de las instrucciones READ, DATA y RESTORE en el programa.

A la línea 80 se llega solamente si coincidían ambos nombres; de ser así, se imprimirá el número de teléfono.

Unas cuantas precisiones. A efectos del ordenador da lo mismo que los valores del DATA vayan en una única línea o en más, y que sean o no contiguos. Cuando ejecute un READ y se acaben los datos del DATA que estuviera leyendo busca la siguiente línea DATA y trabaja con ella. El RESTORE, por su parte, lo que hace es obligar a que el ordenador tome la primera línea de programa donde aparezca un DATA como línea de trabajo para los READ a partir de ese momento.

Instrucciones de control

La cuarta familia de instrucciones (ver capítulo 3) comprende aquellas que modifican el orden de ejecución de las líneas de un programa. El ordenador no está obligado a ejecutar siempre sucesivamente y en el mismo modo las líneas del programa y, por consiguiente, puede adaptarse a situaciones imprevistas.

Es casi superfluo destacar que se trata de las instrucciones, desde el punto de vista lógico, más importantes (y delicadas). El arte de la programación depende en gran parte de la capacidad de servirse de saltos y bucles (cálculos iterativos). Las instrucciones de este grupo son las siguientes:

- salto incodicional GOTO
- salto condicional IF/THEN (ELSE)
- bucle repetitivo FOR/ NEXT

En este grupo se podrían incluir también la instrucción de salto a un subprograma o subrutina GOSUB, y las de salto según un valor (ON GOTO y ON GOSUB), pero de ellas hablaremos en el siguiente volumen de la B.B.I. dedicado al BASIC.

Salto incondicional GOTO

Esta instrucción hace que la ejecución del programa prosiga a partir de la línea que le indiquemos y no de la in-

mediatamente sucesiva. Por ejemplo, al final de un programa podemos escribir GOTO a su línea de comienzo para volver a ejecutarlo desde el principio.

```
10 PRINT "CALCULO DE UN PRODUCTO"
20 INPUT "DAR DOS NUMEROS";A,B
30 P=A*B
40 PRINT "EL RESULTADO ES:";P
50 GOTO 10
60 END
```

Sin la línea 50, el programa, después de haber impreso el resultado de un producto, se pararía (END). Por el contrario, GOTO 10 le hace saltar a la línea 10 y repetir el programa. Si en lugar de GOTO 10 ponemos GOTO 20, ya no se volverá a imprimir el mensaje inicial CALCULO DEL PRODUCTO.

Ojo con este uso del GOTO, pues si obligamos a un programa a repetirse de este modo, ya no tendrá un final lógico y por consiguiente, proseguirá ejecuntando multiplicaciones indefinidamente. Para parar el ordenador, será preciso actuar desde fuera del programa y pulsar una de las teclaks especiales, tales como Break, Stop o Control C. Por lo tanto los bucles sin fin que puede originar el GOTO, que por sí mismos no están prohibidos, se utilizan solamente en casos muy especiales.

La figura 3 muestra el diagrama de bloques de este pequeño programa. Se ve como el programa está cerrado en un anillo sin ninguna posibilidad de terminar "de forma espontánea".

Saltos condicionales IF/THEN (ELSE)

La instrucción GOTO es muy rígida y permite efectuar solamente saltos fijos en el programa. Por el contrario, las instrucciones de salto condicional permiten realizar saltos, o ejecutar grupos completos de instrucciones si una cierta condición es verdadera. Veamos un ejemplo:

```
10 INPUT "ESCRIBE UN NUMERO MENOR QUE 9
   ";N
```

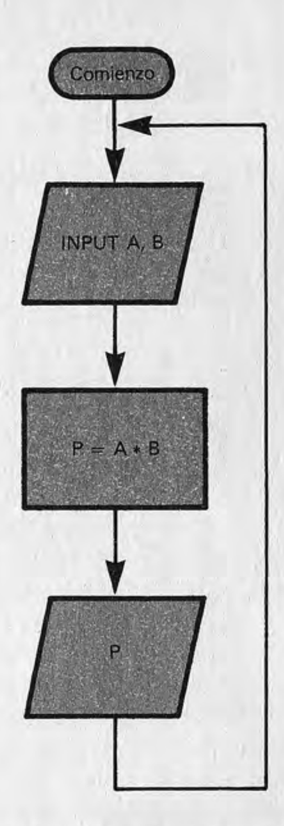



Fig. 3.—Diagrama de bloques de un programa sin salida

```
20 IF N>=9 THEN GOTO 10
30 PRINT "BRAVO,HAS ACERTADO"
```

La instrucción 20 se lee como sigue: si N es mayor o igual que 9, pasar a la línea 10. Si tecleamos, por ejemplo, 15, al ser mayor que 9 la "condición" será verdadera y se producirá el salto del programa.

A modo de inciso, recordamos que el símbolo > quiere decir "mayor que" mientras que >=significa "mayor o igual que". El símbolo < quiere decir "menor que" y el <= significa "menor o igual que".

Por el contrario, si tecleamos un número menor que 9, tal como 5, será falsa la condición y el programa sigue en la línea 30 en donde se imprimirá el mensaje "BRAVO, HAS

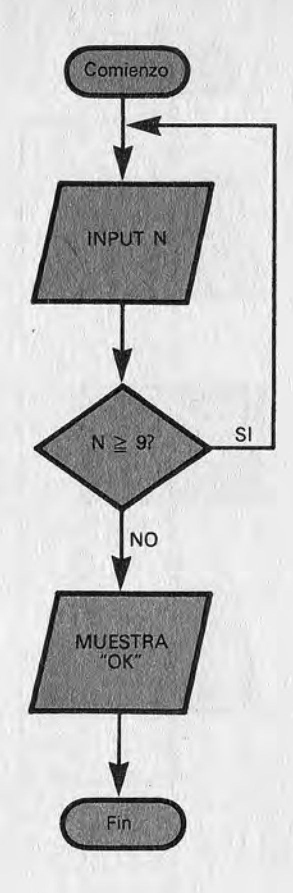


Fig. 4.—Diagrama de bloques de un programa con salto condicional.

ACERTADO". En la Figura 4 se ilustra el diagrama de bloques correspondiente.

Son poquísimas las versiones del BASIC que permiten solamente el IF... THEN GOTO (que además suelen expresar IF... GOTO). La mayor parte permiten una forma más flexible que admite no solamente el salto condicional, sino también la *ejecución condicional* de una o más instrucciones. Veamos un ejemplo:

```
10 INPUT " DAME UN NUMERO";N
20 Q=N*N:REM CALCULO DEL CUADRADO
```



```
30 PRINT "CUAL ES EL CUADRADO DE ";N
40 INPUT X:REM TENTATIVA
50 IF X<Q THEN PRINT "DEMASIADO BAJO!":
   GOTO 30
60 IF X>Q THEN PRINT "DEMASIADO ALTO!":
   GOTO 30
70 PRINT "HAS ACERTADO,ERA ";Q
```

El programa pide un número (N) y calcula su cuadro (Q). A continuación, pide otro número (X), que representa una tentativa de adivinar el cuadrado de N.

En este punto (línea 50) se comprueba si el punto introducido es menor que el valor a adivinar. Si lo fuera, ENTONCES (en inglés THEN) se ejecutará todo el resto de la línea 50. Si no lo fuera, se saltará el resto de la línea y proseguirá con la siguiente línea (60).

Lo anterior se denomina *ejecución condicional* de una instrucción (en nuestro caso, de dos instrucciones, una de ellas PRINT y la otra GOTO), por cuanto que las instrucciones se ejecutan solamente si la condición es verdadera.

En la línea 50 después del THEN, hay dos instrucciones, separadas por el símbolo ":" (dos puntos). En el lenguaje BASIC, las instrucciones separadas por dos puntos se ejecutan una después de la otra, como si estuvieran en líneas sucesivas (casi cualquier versión del lenguaje BASIC lo permite, aunque alguna emplea un símbolo diferente). Habida cuenta de que ambas instrucciones se encuentran después del THEN en la misma línea, se ejecutarán ambas (si la condición es verdadera) o no se ejecutará ninguna (si la condición es falsa).

El efecto sobre el programa ejemplo es que, si el número introducido (X) es menor que la solución (Q), se imprimirá un "DEMASIADO BAJO!" y se ejecutará una instrucción GOTO 30, que solicita el valor de una nueva tentativa. Si el número no es menor que la solución (por consiguiente es igual o mayor), el programa ignorará completamente el resto de la línea 50.

En la línea 60 todo es como antes con la diferencia de que lo que se comprueba es si el número es mayor que el resultado. Si fuera así, se producirá el mensaje correspondiente y el retorno a la línea 30.

Finalmente, si el número no es mayor ni menor que el

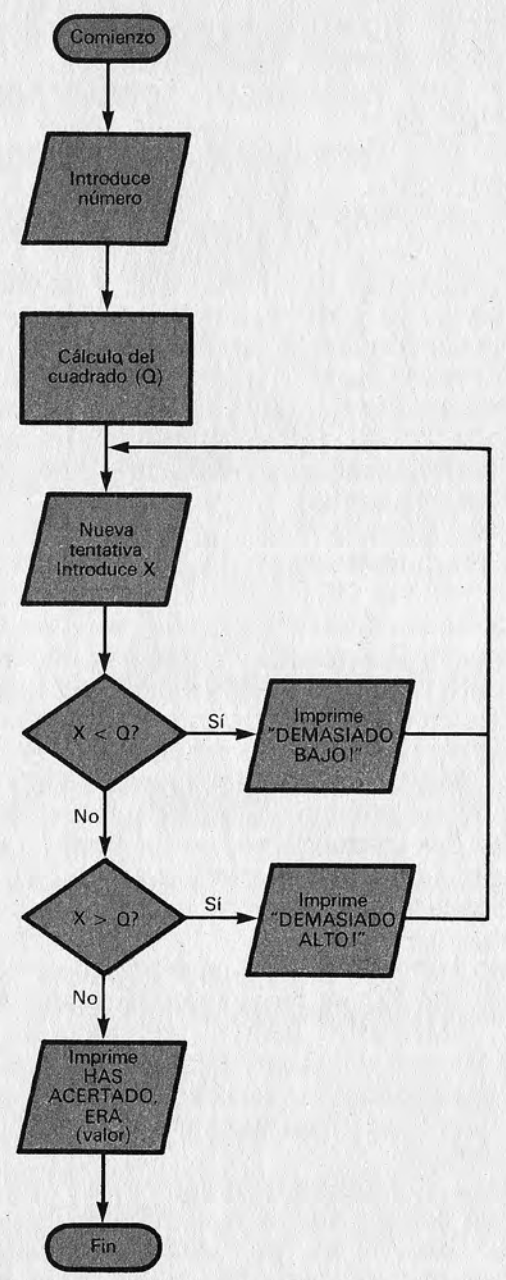


Fig. 5.—Diagrama de bloques de "Adivina el cuadrado".

resultado, deberá ser igual y la respuesta es exacta. En este caso, se generará el mensaje y finalizará el programa (línea 70).

Observe como la misma instrucción PRINT puede imprimir, por ejemplo, una cadena (encerrada entre comillas) e inmediatamente (con punto y coma) el valor de una variable numérica.

La figura 5 ilustra el diagrama de flujo de este programa. Determinadas versiones de BASIC permiten que en lugar de escribir, por ejemplo, IF N> = 9 THEN GOTO 10 se ponga IF N> = 9 THEN 10 o bien IF N> = 9 GOTO 10.

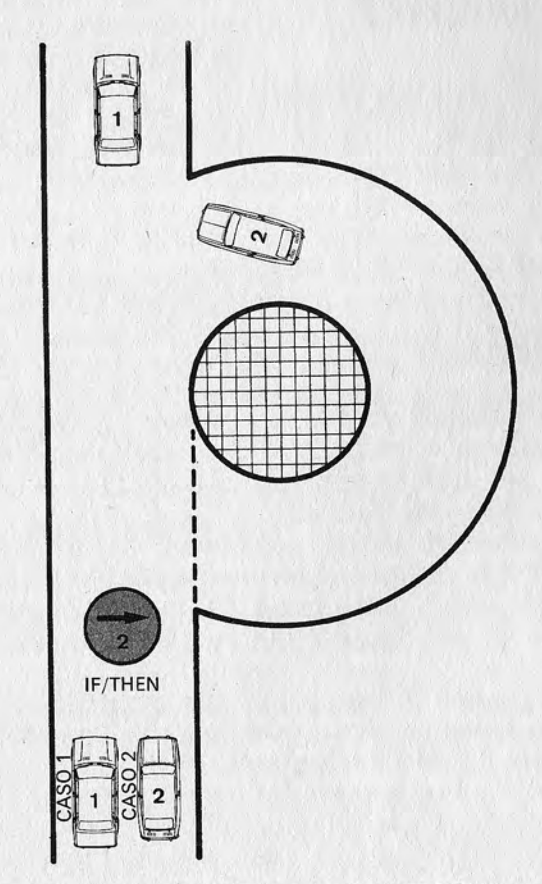


Fig. 6.—Analogía de la instrucción IF/THEN. Si se cumple la condición (ser camión) ejecutará esa línea, pero si no se cumple la saltará.

En muchos ordenadores se admite una extensión del IF/THEN: el IF/THEN/ELSE muy parecido al anterior. Si la condición es cierta el programa realiza las instrucciones que encuentra antes del ELSE y luego salta las instrucciones de éste. Si es falsa, las instrucciones ejecutadas son las del ELSE y el programa continua después normalmente.

Veamos un ejemplo:

```
100 INPUT "DAME TU CLAVE";CL
110 IF (CL=327) THEN PRINT "CLAVE
    CORRECTA":GOTO 200 ELSE PRINT "NO
    ES ESA":END
200 GOTO 200

RUN
DAME TU CLAVE? 134
NO ES ESA
Ok
```

Si la clave es correcta el programa lo dice y continua. En otro caso, tras avisarlo, se acaba.

Expresiones lógicas

Hemos hablado ya de expresiones matemáticas y de operaciones con cadenas. En BASIC existen también "expresiones lógicas" que se emplean sobre todo en las instrucciones condicionales IF/THEN.

Vimos como en las instrucciones IF/THEN la(s) sentencia(s) a ejecutar dependen de una condición. Con gran frecuencia, esta condición puede ser una expresión lógica completa, cuyo resultado se utiliza a fin de realizar la elección.

Una expresión lógica, como una expresión algebraica normal, está constituida por "operandos" y "operadores". Por ejemplo, en la expresión algebraica A + 3, A y 3 son los operandos y "+" es el operador. En las expresiones lógicas, el resultado puede tomar solamente los valores verdadero o falso. Los operadores en las expresiones lógicas del BASIC son los operadores de relación y los operadores lógicos.

Los primeros (de relación) son operadores, vistos ya

con anterioridad, que nos informan de si dos números son entre si iguales o uno es menor, o mayor que el otro. Por ejemplo 5>3 es una expresión verdadera, mientras que es falsa 5<=10. La expresión con variables A<=5 puede ser verdadera o falsa, según el valor tomado por A.

En los ordenadores se pueden emplear los operadores de relación también con las cadenas. Así podemos escribir la expresión B$ = P$; su valor será verdadero si las dos cadenas son iguales y falso en caso contrario (no ha de confundirse con la instrucción de asignación LET B$=P$ que, aunque pueda escribirse B$=P$ constituye por sí misma una instrucción completa, en tanto que el operador relacional se usa durante el curso de una decisión —por ejemplo en una instrucción IF/THEN).

De modo similar, se pueden escribir también expresiones tales como: A$<B$ o C$<=D$. En tal caso, se comprueba si una cadena precede a otra alfabéticamente (como en un diccionario). Del mismo modo que PEDRITO "es mayor" que CARLOS porque le sigue en orden alfabético, se pueden calcular expresiones de relación entre cadenas cualesquiera.

El criterio que permite poner en orden cadenas que incluyan cualquier carácter es el que se deduce del código ASCII. Así, la cadena "2A" precede a "ABC" porque en el código ASCII el 2 está codificado antes que la letra A. Para los datos alfabéticos se sigue, en la práctica, el criterio adoptado por todos los vocabularios.

Los *operadores lógicos* trabajan con valores lógicos y proporcionan un resultado que también lo es. Los operadores de uso más frecuente en el BASIC son AND, OR y NOT.

- AND es el operador del producto lógico. A AND B es una expresión verdadera solamente si A y B son ambos verdaderos y en los demás casos será falsa.
- OR es la suma lógica. A OR B da el resultado de verdadero si es verdadero A, o B, o ambos a la vez. Proporcionará el resultado de falso si A y B son, a la vez, falsos.
- NOT es la negación lógica. NOT A invierte el valor lógico de A de verdadero a falso o viceversa.

Veamos un pequeño ejemplo para comprender cómo se utilizan las expresiones lógicas en un programa lo podemos hacer pidiendo al ordenador que nos indique los nú-

meros comprendidos entre 10 y 20, elegidos entre aquellos que damos libremente a la entrada:

```
10 INPUT " DAME UN NUMERO CUALQUIERA";N
20 IF (N>=10) AND (N<=20) THEN GOTO 50
30 PRINT "ESE NUMERO NO ESTA ENTRE 10 Y
   20"
40 GOTO 10
50 PRINT " ESE NUMERO ESTA ENTRE 10 Y
   20"
60 GOTO 10
```

Instrucciones de bucle: FOR/NEXT

Una de las situaciones más frecuentes que se nos presentan al escribir un programa es aquella en la que debemos repetir muchas veces un conjunto de instrucciones. Vimos ya anteriormente la "realidad" de este problema cuando queríamos cargar los datos en un vector o bien extraer "al contrario" los caracteres de una cadena.

Supongamos que en un vector A$ de 100 elementos se tengan que cargar 100 nombres. No es admisible escribir 100 instrucciones INPUT. Por ello, el BASIC pone a nuestra disposición el par de instrucciones FOR/NEXT que resuelve el problema con una sola instrucción INPUT. Veamos como lo hace

```
10 DIM A$(100)
20 FOR K=1 TO 100
30 NEXT K
```

La instrucción 10 la conocemos ya y es la que dimensiona a 100 el vector de cadenas A$. Las instrucciones 20, 30 y 40 se leen como sigue: para K variando desde 1 a 100 (1, 2, 3, etc), ejecuta la instrucción INPUT del elemento K-ésimo del vector A$.

La instrucción FOR tiene una variable y tres parámetros que actúan sobre ella. El primero es el valor inicial del bucle y de la variable por tanto, (en nuestro caso K=1), el segundo es el extremo del bucle (en el ejemplo dado, 100) y el tercero (que aquí no hemos utilizado, pero que se vio en

el ejemplo incluido al final del capítulo 4), es el paso o incremento del bucle (STEP); cuando no se explicita, el BASIC hace STEP=1.

En lugar de incrementar K de uno en uno se podría incrementar en un valor predeterminado. Por ejemplo, si quisiéramos cargar solamente los elementos de posición impar de A$, podríamos cambiar así la instrucción 20:

```
20 FOR K=1 TO 100 STEP 2
```

En tal caso, K tomaría sucesivamente los valores 1, 3, 5, etc. Si el parámetro del paso STEP no se utiliza, la variable de bucle (K en este ejemplo) se incrementará en uno.

Es posible también recorrer un bucle al revés (vea, de nuevo, el ejemplo final del Capítulo 4). En tal caso, el parámetro STEP deberá tomar un valor negativo.

Cargemos nuestro vector de nombres (A$) al revés.

```
10 DIM A$(100)
20 FOR K=100 TO 1 STEP -1
30 INPUT A$(K)
40 NEXT K
```

La variable de bucle K toma los valores 100, 99, 98, etc, hasta 1.

Las instrucciones de bucle FOR/NEXT pueden ponerse una dentro de otra. Este modo de utilizarlas se denomina, en la jerga «anidamiento de bucles». Ponemos de manifiesto el anidamiento de dos bucles, con un ejemplo, muy sencillo pero eficaz, que es la elaboración de una tabla pitagórica.

```
5 REM  ***********************
10 REM *  TABLA PITAGORICA   *
15 REM ***********************
20 FOR I=1 TO 10
30 FOR J=1 TO 10
40 PRINT I*J;" ";
50 NEXT J
60 PRINT
70 PRINT
80 NEXT I
```

Las instrucciones 30, 40, y 50, constituyen el bucle interior, que calcula e imprime una línea de la tabla con el índice I, que se mantiene constante, y el índice J que varía desde 1 hasta 10. Observe que la instrucción PRINT en la línea 40 termina con un punto y coma (;) para no saltar una línea durante la escritura de una fila.

Para cada línea se repite el mismo ciclo con un valor diferente de I. El bucle exterior (en el que varía I) hace repetir diez veces todo el bucle interior (en el que varía J). La instrucción PRINT en la línea 60 sirve para saltar una línea al final de cada una de ellas. Probemos a ejecutar el programa y se obtendrá el resultado ilustrado en la Figura 7.

¿Por qué la tabla no está exactamente alineada? Dejamos al lector encontrar la respuesta, con la ayuda de la instrucción:

```
45 IF I*J<10 THEN PRINT " ";
```

```
RUN
1  2  3  4  5  6  7  8  9  10
2  4  6  8  10  12  14  16  18  20
3  6  9  12  15  18  21  24  27  30
4  8  12  16  20  24  28  32  36  40
5  10  15  20  25  30  35  40  45  50
6  12  18  24  30  36  42  48  54  60
7  14  21  28  35  42  49  56  63  70
8  16  24  32  40  48  56  64  72  80
9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
Ok
```

Fig. 7.—Cómo aparece la tabla de números generada por el programa "Tabla pitagórica"

que pone todo en su lugar, como se ve en el siguiente listado y ejecución.

Para que un anidamiento de bucles sea correcto es imprescindible que ningún NEXT se encuentre entre dos FOR sin completar (sin llegar a su NEXT). Por tanto el orden en que se encuentren los NEXT debe coincidir con el que usamos para escribir los FOR (ver Figura 8).

Un programa-resumen: «Archivo con punteros»

Para completar este primer volumen sobre el BASIC, consideramos oportuno proponerles un programa un poco más completo que los habituales ejemplos que hemos ido viendo y que, si se enriquece con las instrucciones para el empleo de casete o de disco flexible, puede resultar verdaderamente útil. Se trata de un archivo de nombres que los mantiene automáticamente en orden alfabético.

En los capítulos anteriores, no tuvimos ocasión de hablar de archivos o ficheros (lo trataremos en el segundo volumen del BASIC) ni de sus operaciones normales de control como, por ejemplo, ordenar sus elementos por orden al-

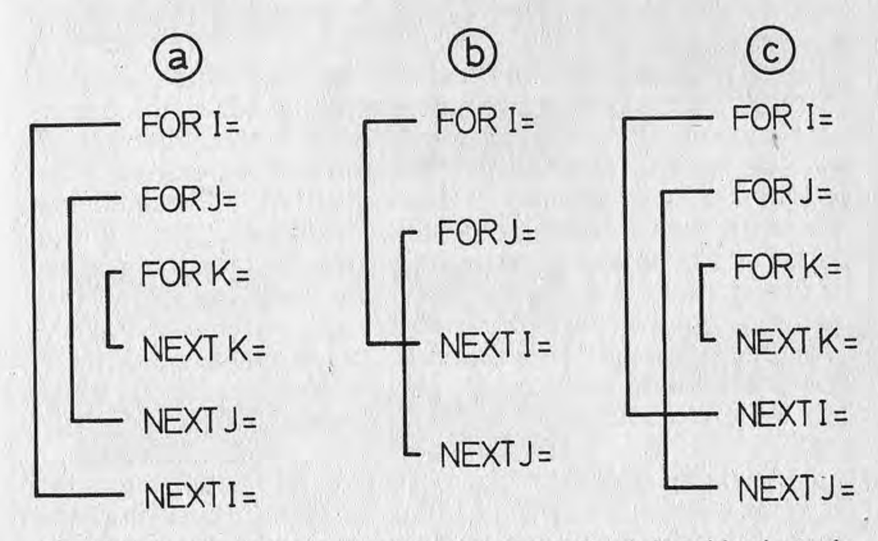


Fig. 8.—Ejemplo correcto (a) e incorrectos (b, c) de anidamiento de bucles.

fabético, añadirles otros nuevos, anular alguno de los antiguos, etc.

Pues bien, el programa que les proponemos pretende, precisamente, actuar como un archivo, con sus características instrucciones de control.

El primer problema que se presenta, al tratar los archivos, es el de ordenarlos (en inglés, «sort»). El orden más simple que se puede considerar es el alfabético, pero vimos ya, al hablar de los operadores relacionales, que este concepto puede extenderse también a otros caracteres tomando como referencia el código ASCII. De este modo, se pueden poner en orden los códigos de los productos de una empresa, tanto los que contengan letras como los que posean dígitos u otros caracteres.

No queremos adentrarnos ahora en las técnicas de ordenación, pero se puede intuir con facilidad que para ordenar una serie de datos es necesario que el ordenador gaste, a veces, mucho tiempo (indudablemente, subirá su valor con el del número de datos a ordenar). Uno de los criterios más sencillos de ordenación, por ejemplo, procede de manera repetitiva, extrayendo el elemento más pequeño (o más grande) de toda la serie, luego el menor (o mayor) de los que quedan y así sucesivamente. Al decir extraer queremos significar que toma su valor y lo escribe en otra parte.

Ahora bien, estas operaciones de reescritura pueden ser muy costosas; piense, por ejemplo en los datos de archivos complejos como los de un registro civil. Mientras se trate de realizar ejercicios en un ordenador personal y se ordenen pocos registros, (así se llaman en informática los elementos de un archivo o fichero) todos los algoritmos van bien, pero cuando entremos en trabajos «comprometidos», es preciso estar muy atentos a ahorrar tiempo evitando, por ejemplo, inútiles reescrituras de datos.

El ejemplo que proponemos crea, y mantiene ordenado, un fichero de nombres sin reescribir nunca ningún nombre, teniendo solamente en cuenta una sucesión de punteros.

Los punteros son un tipo particular de variables que relacionan entre sí los elementos de una lista. Contienen siempre el valor de una dirección, de ahí su nombre, pues «apuntan» hacia esa dirección. Con este sistema el primer elemento contiene, además de sus datos, un puntero dirigido al segundo elemento; éste contiene otro puntero dirigido al tercero y así sucesivamente. En la Figura 9 vemos cómo se presenta una lista de elementos vinculados entre sí por punteros y como entrará entre el primer y segundo elementos de la lista otro nuevo.

Recordemos que ahora cada elemento tiene dos contenidos: el dato y su puntero, que señala al elemento siguiente.

La utilidad de los punteros se comprende inmediatamente considerando, por ejemplo, que para insertar un nuevo elemento en la lista basta con adaptar los punteros (Figura 9).

El programa «Archivo» (Figura 10) puede desempeñar tres funciones: insertar nuevos elementos (I), leer los nombres (L) y leer, además de los nombres, sus punteros (P).

El archivo está contenido en los vectores N$ y P, dimensionados en la línea 140 a 50, pero podría tener cualquier longitud o, mejor todavía, estar grabado en casete o disco.

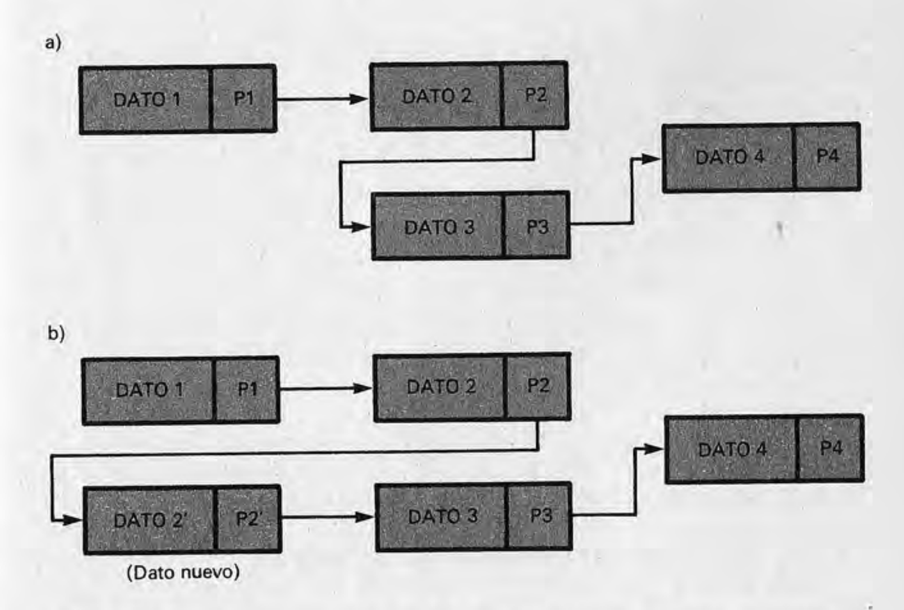


Fig. 9.—Funcionamiento de los punteros (a) y ejemplo de inserción en una lista (b).

```
100 REM ***********************************
110 REM *   ARCHIVO ORDENADO CON PUNTEROS   *
120 REM ***********************************
130 REM
140 DIM N$(50),P(50)
150 REM
160 PL=2:REM PRIMER PUNTERO LIBRE
170 REM
180 PRINT :PRINT "QUE OPERACION DESEA?"
200 PRINT "   INSERCIONES          <I>  "
210 PRINT "   LEER NOMBRES         <L>  "
220 PRINT "   NOMBRE Y PUNTERO     <P>  "
230 REM
240 INPUT R$
250 IF R$="I" THEN GOTO 400
260 IF R$="L" THEN GOTO 600
270 IF R$="P" THEN GOTO 800
280 GOTO 240
290 REM
400 REM *************************
401 REM *   INSERCION DE NOMBRES   *
402 REM *************************
410 REM
415 PRINT
420 INPUT "NOMBRE";N$:IF N$="" THEN GOTO 180
430 PR =P(1)
440 PE=1:REM PE=PUNTERO DE EXPLORACION
450 IF PR=0 THEN GOTO 530
470 REM
480 IF N$(PR)>N$ THEN GOTO 530
490 PE=PR
500 PR=P(PR)
510 GOTO 450
520 REM
530 P(PL)=PR
540 P(PE)=PL
550 N$(PL)=N$
560 PL=PL+1
570 GOTO 420
580 REM
600 REM ************************
601 REM *   LECTURA DE NOMBRES   *
602 REM ************************
610 REM
```

Fig. 10.—Programa "Archivo" y ejecución.


```
620 PRINT
630 PR=P(1)
640 REM
650 IF PR =0 THEN GOTO 180
660 PRINT N$(PR)
670 PR=P(PR)
680 GOTO 650
690 REM
800 REM ***********************************
801 REM *   LECTURA DE NOMBRES Y PUNTEROS   *
802 REM ***********************************
810 REM
820 PRINT
830 FOR K=1 TO PL
840 PRINT K,N$(K),P(K)
850 NEXT K
860 GOTO 180
```

```
RUN

QUE OPERACION DESEA?
   INSERCIONES          <I>
   LEER NOMBRES         <L>
   NOMBRE Y PUNTERO     <P>
? I

NOMBRE? FERNANDO
NOMBRE? CARLOS
NOMBRE? EMILIO
NOMBRE? ANTONIO
NOMBRE? VICENTE
NOMBRE? MARIANO
NOMBRE? ENRIQUE
NOMBRE?

QUE OPERACION DESEA?
   INSERCIONES          <I>
   LEER NOMBRES         <L>
   NOMBRE Y PUNTERO     <P>
? L
```

Fig. 10.—Continuación del programa "Archivo".

```
ANTONIO
CARLOS
EMILIO
ENRIQUE
FERNANDO
MARIANO
VICENTE

QUE OPERACION DESEA?
    INSERCIONES              <I>
    LEER NOMBRES             <L>
    NOMBRE Y PUNTERO         <P>
?P

 1                                5
 2             FERNANDO           7
 3             CARLOS             4
 4             EMILIO             8
 5             ANTONIO            3
 6             VICENTE            0
 7             MARIANO            6
 8             ENRIQUE            2
 9                                0

QUE OPERACION DESEA?
    INSERCIONES              <I>
    LEER NOMBRES             <L>
    NOMBRE Y PUNTERO         <P>
?
```

Fig. 10.—Continuación del programa "Archivo".

El vector N$ contiene los nombres, mientras que el vector P contiene los punteros correspondientes. Lo que hay que tener bien presente es que los nombres no se desplazarán nunca en el array cuando se inserte alguno nuevo y se haga la ordenación; serán únicamente los punteros los que se modifiquen.

Para comprender cómo se comportan los punteros, observe primero cómo se produce la lectura del archivo ordenado (opción L del menú) en las líneas del programa desde la 600 a la 690. Se comienza por leer el puntero P (1) en la línea 630; si es diferente de cero (final del archivo) se lee el nombre N$ (PR), y se toma el puntero correspondiente P(PR) que nos servirá para localizar el nombre siguiente y así sucesivamente.

Las líneas del programa desde la 40 a 580 crean el archivo insertando nuevos elementos y ajustando los punteros de modo que los nombres sean legibles en orden alfabético. La Figura 11 muestra cómo se controlan los punteros.

Después de haber introducido los nombres ALDO y CARLOS, por casualidad ya ordenados, para introducir AN-

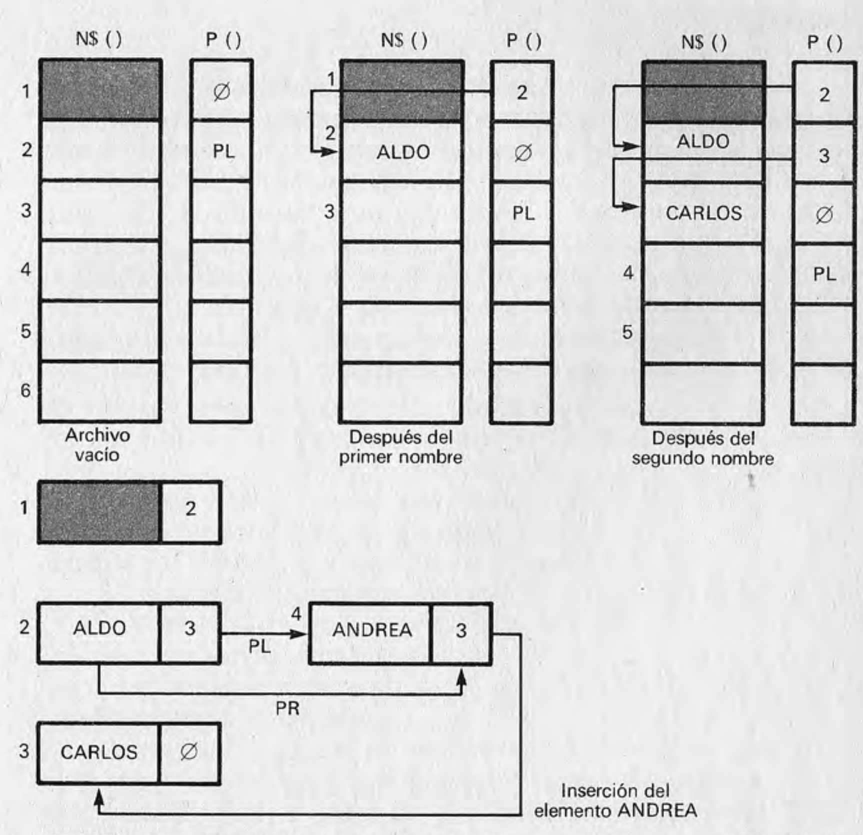


Fig. 11.—Funcionamiento de los punteros del programa "Archivo". PL es el puntero del elemento libre, PR es el puntero real (hacia adelante) y PE es la variable para la exploración en el interior del archivo.

DREA se debe leer en el PL cual es el próximo elemento libre (4) y actualizar los punteros de los elementos 2 (ALDO) y 4 (CARLOS).

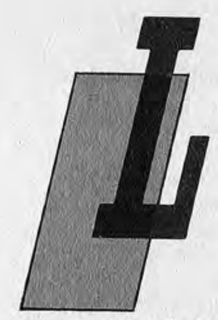
Observe que la única instrucción que compara el orden de dos nombres está en la línea 480, pero no se realiza ningún desplazamiento, sino una actualización de punteros (líneas 490 y 500) desarrollada de una ingeniosa manera. Para la opción P, como verá, usamos un bucle FOR/NEXT. Esto es debido a que aquí no nos interesa sacar los nombres por orden alfabético, sino tan sólo saber los nombres que hay y sus punteros respectivos.

Despedida

Este primer volumen sobre el lenguaje BASIC se le habrá pasado muy rápidamente, en armonía con la sencillez que caracteriza el lenguaje. Llegados a este punto, podemos tener la seguridad de que vale la pena continuar este recorrido por el mundo del BASIC, leyendo también el siguiente volumen que nuestra Biblioteca Básica Informática le dedicará. Después de haberlo hecho, podremos afrontar metas más difíciles, ayudados además por el resto de los volúmenes de nuestra colección, que irán apareciendo «sin prisas, pero sin pausas». Les deseamos un provechoso estudio.

APENDICE A

EL CODIGO ASCII



Los datos que un ordenador procesa y almacena en su interior suelen ser datos numéricos o cadenas de caracteres. El tratamiento de los números no es difícil, por cuanto que el ordenador es una máquina electrónica digital y, por consiguiente, puede manejar fácilmente los números binarios (constituidos solamente por los dígitos 0 y 1). Ya que todos los números, incluso los decimales comunes, se pueden convertir fácilmente en forma binaria (ver el número 1 de la B. B. I.) siempre será posible almacenarlos en los ordenadores. Evidentemente, en cualquier ordenador personal estas operaciones de transformación en código binario se realizan de forma automática, por lo que quien programa no debe preocuparse en absoluto de ellas.

Almacenar caracteres (letras, signos de puntuación, dígitos, etc) será pues bastante fácil si antes los hemos condificado adecuadamente. A cada carácter, letra, etc, hacemos corresponder un número binario que se almacenará luego como los demás.

Las posibilidades de relacionar números y caracteres son muy numerosas. La más frecuentemente utilizada en los ordenadores personales es la codificación americana ASCII. La palabra ASCII es la abreviatura de «American Standard Code for Information Interchange», que significa «Código americano para el intercambio de información». El código

ASCII, aceptado universalmente no sólo en los ordenadores personales, permite codificar 128 caracteres. A menudo se extiende a 256 caracteres para utilizar un byte completo (un byte está constituido por 8 bits y $2^8=256$), lo que permite incluir algunos caracteres pseudográficos o de control adicionales.

La tabla siguiente indica los 128 primeros caracteres del código ASCII estándar (incluido el cero).

CODIGO ASCII

CARACTER	CODIGO			DEFINICION
	BINARIO	DECIMAL	HEXADECIMAL	
NUL	0000 0000	0	00	Nulo
SOH	0000 0001	1	01	Principio de encabezamiento
STX	0000 0010	2	02	Comienzo de texto
ETX	0000 0011	3	03	Fin de texto
EOT	0000 0100	4	04	Fin de transmisión
ENQ	0000 0101	5	05	Pregunta
ACK	0000 0110	6	06	Acuse de recibo
BEL	0000 0111	7	07	Timbre (señal)
BS	0000 1000	8	08	Retroceso
HT	0000 1001	9	09	Tabulación horizontal
LF	0000 1010	10	0A	Cambio de renglón
VT	0000 1011	11	0B	Tabulación horizontal
FF	0000 1100	12	0C	Página siguiente
CR	0000 1101	13	0D	Retorno de carro
SO	0000 1110	14	0E	Fuera de código
SI	0000 1111	15	0F	En código
DLE	0001 0000	16	10	Encaje de transmisión
DC1	0001 0001	17	11	Mando de dispositivo auxiliar 1
DC2	0001 0010	18	12	Mando de dispositivo auxiliar 2
DC3	0001 0011	19	13	Mando de dispositivo auxiliar 3
DC4	0001 0100	20	14	Mando de dispositivo auxiliar 4
NAK	0001 0101	21	15	Acuse de recibo negativo
SYN	0001 0110	22	16	Sincronización
ETB	0001 0111	23	17	Fin de bloque de transmisión
CAN	0001 1000	24	18	Cancelación
EM	0001 1001	25	19	Fin de medio físico

CODIGO ASCII

CARACTER	CODIGO			DEFINICION
	BINARIO	DECIMAL	HEXADECIMAL	
SUB	0001 1010	26	1A	Sustitución
ESC	0001 1011	27	1B	Escape
FS	0001 1100	28	1C	Separador de fichero
GS	0001 1101	29	1D	Separador de grupo
RS	0001 1110	30	1E	Separador de registro
US	0001 1111	31	1F	Separador de unidad
	0010 0000	32	20	Espacio en blanco
!	0010 0001	33	21	Admiración
"	0010 0010	34	22	Comillas
#	0010 0011	35	23	Símbolo número (cancela)
$	0010 0100	36	24	Símbolo dólar
%	0010 0101	37	25	Porcentaje
&	0010 0110	38	26	"Ampersand"
'	0010 0111	39	27	Acento
(	0010 1000	40	28	Apertura de paréntesis
)	0010 1001	41	29	Cierre de paréntesis
*	0010 1010	42	2A	Asterisco
+	0010 1011	43	2B	Signo más
,	0010 1100	44	2C	Coma
–	0010 1101	45	2D	Guión (signo menos)
.	0010 1110	46	2E	Punto
/	0010 1111	47	2F	Símbolo división ("slash")
0	0011 0000	48	30	
1	0011 0001	49	31	
2	0011 0010	50	32	
3	0011 0011	51	33	
4	0011 0100	52	34	
5	0011 0101	53	35	
6	0011 0110	54	36	
7	0011 0111	55	37	
8	0011 1000	56	38	
9	0011 1001	57	39	
:	0011 1010	58	3A	Dos puntos
;	0011 1011	59	3B	Punto y coma
<	0011 1100	60	3C	Menor que
=	0011 1101	61	3D	Igual
>	0011 1110	62	3E	Mayor que
?	0011 1111	63	3F	Interrogante
@	0100 0000	64	40	"Atpersand", arroba
A	0100 0001	65	41	
B	0100 0010	66	42	
C	0100 0011	67	43	
D	0100 0100	68	44	
E	0100 0101	69	45	

CODIGO ASCII

CARACTER	CODIGO BINARIO	CODIGO DECIMAL	CODIGO HEXADECIMAL	DEFINICION
F	0100 0110	70	46	
G	0100 0111	71	47	
H	0100 1000	72	48	
I	0100 1001	73	49	
J	0100 1010	74	4A	
K	0100 1011	75	4B	
L	0100 1100	76	4C	
M	0100 1101	77	4D	
N	0100 1110	78	4E	
O	0100 1111	79	4F	
P	0101 0000	80	50	
Q	0101 0001	81	51	
R	0101 0010	82	52	
S	0101 0011	83	53	
T	0101 0100	84	54	
U	0101 0101	85	55	
V	0101 0110	86	56	
W	0101 0111	87	57	
X	0101 1000	88	58	
Y	0101 1001	89	59	
Z	0101 1010	90	5A	
[	0101 1011	91	5B	Apertura de corchete
\	0101 1100	92	5C	Barra invertida ("Back slash")
]	0101 1101	93	5D	Cierre de corchete
ˆ	0101 1110	94	5E	Acento Circunflejo
—	0101 1111	95	5F	Güión de subrayado
`	0110 0000	96	60	Acento inverso
a	0110 0001	97	61	
b	0110 0010	98	62	
c	0110 0011	99	63	
d	0110 0100	100	64	
e	0110 0101	101	65	
f	0110 0110	102	66	
g	0110 0111	103	67	
h	0110 1000	104	68	
i	0110 1001	105	69	
j	0110 1010	106	6A	
k	0110 1011	107	6B	
l	0110 1100	108	6C	
m	0110 1101	109	6D	
n	0110 1110	110	6E	
o	0110 1111	111	6F	
p	0111 0000	112	70	

CODIGO ASCII

CARACTER	CODIGO BINARIO	CODIGO DECIMAL	CODIGO HEXADECIMAL	DEFINICION
q	0111 0001	113	71	
r	0111 0010	114	72	
s	0111 0011	115	73	
t	0111 0100	116	74	
u	0111 0101	117	75	
v	0111 0110	118	76	
w	0111 0111	119	77	
x	0111 1000	120	78	
y	0111 1001	121	79	
z	0111 1010	122	7A	
<	0111 1011	123	7B	Apertura de corchete
|	0111 1100	124	7C	Barra vertical
>	0111 1101	125	7D	Cierre de corchete
—	0111 1110	126	7E	Tilde
DEL	0111 1111	127	7F	Borrado, supresión

APENDICE B

RESUMEN DE LAS INSTRUCCIONES ESTUDIADAS



ara facilitar al lector la rápida solución de cualquier duda sobre la sintaxis de las instrucciones vistas hemos confeccionado un pequeño resumen-diccionario, que esperamos les resulte útil.

Además de definir la función de la instrucción y su sintaxis incluye unos cuantos ejemplos de muestra. En ningún caso incluimos el número de línea, pues damos por sentado que el lector sabe que si la instrucción forma parte de un programa debe llevarlo y, en caso contrario (ejecución directa), no.

En las definiciones que siguen usaremos los siguientes símbolos:

[ ] elemento opcional
{ } posibilidades alternativas
<> deben ser determinados por el usuario.

ASC

Devuelve el código ASCII correspondiente al carácter dado. Si fuera una cadena da el del primer carácter.

ASC ({<variable alfanumérica> <"cadena">})

ASC (V$)
ASC ("P")

CHR$

Devuelve el carácter correspondiente al código ASCII dado

CHR$ '(<número>)

CHR$ (65)

DATA

Almacena valores constantes (numéricos o caracteres) para que sean leídos mediante READ

DATA <constante 1> [,<constante 2>...]

DATA "BBI", 1, 1985

DIM

Dimensiona el tamaño máximo de variables con índice

DIM <variable 1> [,<variable 2>...]

DIM NOMBRE$(30,7), PE(8Ø), A(3)

FOR/NEXT

Ejecuta las instrucciones comprendidas entre ambas hasta que la variable de bucle (cuyo valor inicial lo da <expresión 1>) alcanza, incrementándose a cada pasada en escalones dados por <expresión 3>, por defecto 1, el valor de <expresión 2>.

FOR <variable bucle>=<expresión 1> TO <expresión 2> [STEP <expresión 3>] NEXT <variable bucle>

FOR I=(F-1) TO 5 STEP F
NEXT I

Funciones matemáticas

Devuelven el resultado de aplicar la función concreta correspondiente al dato

FUNCION DADA (<dato>)

SQR (8+C)	raíz cuadrada
SIN (0.18*H)	seno (ángulo en radianes)
COS (AMG)	coseno (ángulo en radianes)
TAM (CIRC)	tangente (ángulo en radianes)
ATM (IMV)	arco tangente
LOG (PQS)	logaritmo neperiano
EXP (X)	antilogaritmo neperiano (e^x)

GOTO

Realiza un salto al número de línea dado (o al resultado de la expresión en algunos casos)

GOTO {[<número de línea>] [<expresión>]}

GOTO 300
GOTO INI + 40 (QL y Spectrum)

IF/GOTO

Si se cumple la condición ejecuta el GOTO si no, lo salta.

IF <condición> GOTO {[<número de líneas>] [<expresión>]}

IF M<1 GOTO 500

IF/THEN

Si la condición es cierta realiza el resto de las instrucciones de la línea si no, se las salta

```
IF <condición> THEN <instrucción 1> [: <instrucción 2>...]
```

```
IF M<Ø THEN M=0
```

IF/THEN/ELSE

Si la condición es cierta realiza lo que venga hasta el ELSE y luego salta la instrucción de éste. Si no es cierta realiza lo del ELSE y continua.

```
IF <condición> THEN <bloque instrucciones 1>
[ELSE <bloque de instrucciones 2>]
```

```
IF M<Ø THEN M=Ø ELSE M=2*M
```

INPUT

Asigna los valores introducidos por teclado a las variables correspondientes.

```
INPUT [;] ["<Mensaje>"{;,}] <variable 1> [,<variable 2>...]
```

```
INPUT A, B$, VECT(3,5)
INPUT "JUNTO";X
```

LEFT$

Toma los n caracteres más a la izquierda de la cadena dada.

```
LEFT$ (<cadena>, n)
```

```
B$=LEFT$ ("JOSE",2)
PRINT LEFT$ (B$,1)
```


LEN

Devuelve la longitud (número de caracteres) de la cadena

```
LEN (<cadena>)
```

```
L=LEN ("PEPE")
P=LEM (A$)
```

LET

Asigna a la variable situada a la izquierda del signo = el resultado de la expresión ubicada a la derecha.

```
[LET] <variable>=<expresión>
```

```
LET A=27*C
B=584
C$="JUAN PEREZ"
```

MID$

Toma n caracteres de la cadena, empezando por el número i-ésimo.

```
MID$ (<cadena>, i, n)
```

```
MID$ ("PEPE", 2,2)
MID$ (M$, 8,15)
```

PRINT

Visualiza en la pantalla mensajes y valores de variables u operaciones, es decir, cualquier "expresión"

```
PRINT <expresión 1> {[;] [,]} <expresión 2>...
```

```
PRINT "VOLUMEN ";I;"DE B.B.I.";
PRINT B$,C;I,"VALOR"
```

READ

Asigna a las variables dadas los valores extraídos de sentencias DATA.

```
READ <variable 1> [,<variable 2>... }
```

READ D,H$, PE(1,1)

RESTORE

Lleva el puntero de DATA activo al primer valor de la primera sentencia DATA, o de la que se señale (por su número de línea).

```
RESTORE [{<número de línea> <expresión>}]
```

RESTORE
RESTORE 80
RESTORE I+8

RIGHT$

Toma los n caracteres más a la derecha de la cadena dada.

```
RIGHT$ (<cadena>, n)
```

B$=RIGHT$ ("SAN PEDRO",5)
RIGHT$ (C$,3)

BIBLIOGRAFIA

Programación en BASIC: un método práctico.
Dachslager y Zucker. *Anaya Multimedia.*

Diseño de gráficos y videojuegos. Tratamiento en tres dimensiones.
Angel y Jones, 1985. *Anaya Multimedia.*

Programación avanzada en BASIC.
Bishop. *Anaya Multimedia.*

El libro del IBM, PC, XT, AT.
L. E. Frenzel Jr. / L. E. Frenzel III, 1985. *Anaya Multimedia.*

Diccionario de informática ingles-español-francés.
G. A. Mania, 1985. *Paraninfo.*

Diccionario del BASIC.
Willie Hart, 1985. *Paraninfo.*

Como programar su COMMODORE 64 1 - BASIC, gráficos, sonido.
F. Montell, 1985. *Paraninfo.*

Como programar un COMMODORE 64 2 - Lenguaje máquina, E/S, periféricos.
F. Montell, 1985. *Paraninfo.*

Tratamientos de textos con BASIC.
G. Quaneaux, 1985. *Paraninfo.*

Informática para no avanzados.
G. Willmott, 1985. *Deusto.*

102 programas para su APPLE.
Desconchat. *Elisa.*

Claves para el COMMODORE 64.
D. Gean David. *Elisa.*

COMMODORE 64, para todos.
Boisgontier Brebion Foucault. *Elisa.*

Los ordenadores. Fundamentos y sistemas.
J. C. Giarratano, 1984. *Díaz de Santos.*

Conceptos actuales sobre la tecnología de los ordenadores.
J. C. Giarratano, 1984. *Díaz de Santos.*

Diccionario de informática ingles-español. Glosario de términos informáticos.
Olivetti, 5. Edic, 1984. *Paraninfo.*

Como prograrmar ordenadores personales.
R. Farrando, 1985. *Marcombo.*

Diccionario de informática.
Masson, 2. Edic, 1985. *Masson.*

Glosario de computación.
Alan Freedman, 1984. *McGraw-Hill.*

INDICE GENERAL

15 **Dibujar con el ordenador**
Profundizando en una de las facetas útiles y divertidas que nos ofrecen los ordenadores.

16 **Tratamiento de textos... para escribir con el ordenador**
Cómo convertir su ordenador en una máquina de escribir con memoria y todo tipo de posibilidades.

17 **Diseño de juegos**
Particularidades características de esta aplicación de los ordenadores.

18 **LOGO: la tortuga inteligente**
Un lenguaje conocido por su «cursor gráfico», la tortuga, y sus aplicaciones pedagógicas al alcance de su mano.

19 **BASIC y tratamiento de imágenes**
Todo lo que en **¡Y llegó el BASIC!** no se pudo ver sobre las imágenes y gráficos en el BASIC.

20 **Bancos de datos (I)**
Peculiaridades de una de las aplicaciones de los ordenadores más interesantes, y que más dinero mueven.

21 **Bancos de datos (II)**
Profundizando en sus características.

22 **Paquetes integrados: Lotus 1-2-3 y Simphony**
Estudio de dos de los paquetes integrados (Hoja de cálculo + base de datos + ...) más conocidos.

23 **dBASE II y dBASE III**
Cómo aprovechar las dos versiones más recientes de esta importante base de datos.

24 **Los ordenadores uno a uno**
Un amplio y completo estudio comparativo.

25 **Cálculo numérico en BASIC**
Una aplicación especializada a su disposición.

26 **Multiplan**
Cómo hacer uso de este moderno paquete de aplicaciones.

27 **FORTRAN y COBOL**
Dos lenguajes muy especializados y distintos.

28 **FORTH: anatomía de un lenguaje inteligente**
Principales características de un lenguaje moderno, flexible y de amplio uso, en la robótica.

29 **Cómo realizar nuestro propio banco de datos**
Conocimientos necesarios para poder fabricar un banco de datos a nuestro gusto y medida.

30 **Los paquetes integrados uno a uno**
Todos los que usted puede encontrar en el mercado.

NOTA: Ingelek, S. A. se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de la colección.

NOTAS

*Q*UIZAS el rasgo más característico de un ordenador personal sea el carácter inmediato de su utilización. Esto no significa, como muchas veces se piensa, que su acceso sea simple, sino que permanece siempre a nuestra disposición, atento a nuestras instrucciones. Un buen ordenador personal debe ser "interactivo", es decir, debe invitar al diálogo. Para ello tendrá que disponer de un lenguaje que permita este coloquio; el más sencillo y difundido es, sin duda, el BASIC.

La simplicidad de sus reglas y el carácter interpretativo del lenguaje BASIC hacen que cada instrucción o grupo de instrucciones pueda utilizarse y comprobarse de inmediato.

Después de los anteriores tomos de la colección creemos que es el momento adecuado para dar, de manera optimista y sin temores, nuestros primeros pasos en BASIC.

Insistimos una vez más en que NO es preciso que todos lleguemos a ser programadores expertos, pues hay una ingente cantidad de software disponible y la tarea de programar exige bastante tiempo y paciencia. Sin embargo, una base mínima es siempre útil y nos servirá, al menos, para comprender mejor el funcionamiento de los programas y sacar mayor rendimiento de estos y del ordenador.

375 pts.

El primer programa

Hay dos modos de aprender un lenguaje de programación: estudiar detenidamente todas sus reglas desde la primera a la última y luego, tratar de escribir los primeros programas; o bien, quizá más pragmáticamente, tratar de escribir inmediatamente programas sencillos y aprender las reglas poco a poco. Nosotros adoptaremos este segundo método que, en nuestra opinión, es menos pesado y más eficaz. Veamos un programa en BASIC de lo más facilón:

```
10 INPUT A
20 INPUT B
30 INPUT C
40 PRINT A
50 PRINT B
60 PRINT C
```

¿Qué observamos?

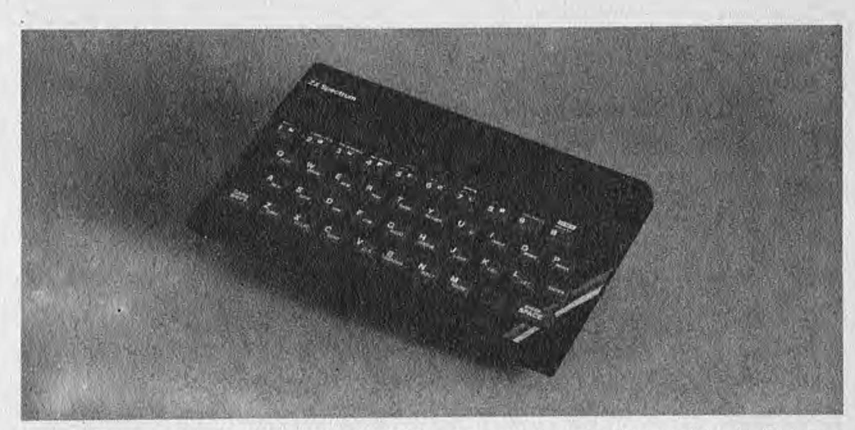
El programa está escrito en varias líneas. Cada línea se inicia con un número que va aumentando de forma sucesiva. Hay también palabras inglesas, como INPUT y PRINT, y letras (A, B, C). Lo que no vemos, pero tiene una enorme importancia, es que cada línea se finaliza pulsando una tecla especial, que equivale a la que en las máquinas de escribir produce el retroceso del carro (en inglés, «Carriage Return»).

Cada línea de cualquier programa debe terminar siempre con la pulsación de esta tecla. Según los ordenadores se llama de «RETURN,» «ENTER,» «CR...». Cuando se utiliza un nuevo ordenador, esta es la primera tecla que se debe localizar.

¿Qué hace este sencillo programa? En las líneas 10,20 y 30 encontramos escrita la palabra INPUT seguida por una letra. INPUT es una instrucción del lenguaje BASIC e indica al ordenador que solicite la entrada de datos.

Las letras A, B y C son tres nombres cualesquiera de variables numéricas, iguales que las que aprendemos a manejar en la escuela. Cuando el programa se ejecuta, el ordenador solicita tres veces seguidas un número, imprimiendo





Fotografía 1.—En estos teclados se pueden ver las teclas de Enter y Return, que realizan la función "Retorno de carro".

cada vez un signo de interrogación y quedando parado hasta que no introduzcamos (terminando con un «RETURN») dicho número por el teclado. El primer valor que metamos se asignará a (se depositará dentro de) la variable A, el segundo a la B y el tercero a la C. Más adelante profundizaremos en el concepto de variable y sus tipos; de momento, lo importante es entender que mediante la instrucción INPUT se pueden comunicar datos al ordenador y asociarlos a variables.

Continuemos con el examen del programa

En las líneas 40, 50 y 60 encontramos otra instrucción BASIC: PRINT. PRINT significa literalmente «imprimir» pero,

por extensión se refiere a la presentación visual en pantalla. Desempeña por tanto la función opuesta a INPUT, puesto que saca los datos fuera del ordenador, presentándoselos al usuario.

PRINT conduce los datos a la pantalla del ordenador; otras instrucciones de salida (que veremos en posteriores capítulos) llevan los datos a la impresora, a la cinta de casete, o a discos flexibles. Al ejecutar el programa, después de haber tecleado los tres datos como respuesta a los signos de interrogación que nos presenta el ordenador veremos aparecer en la pantalla los valores asignados a cada variable. El proceso sería:

RUN	
?12	Pide valor de A.
?23	Pide valor de B.
?34	Pide valor de C.
12	Presenta valor asignado a la variable A.
23	Presenta valor asignado a la variable B.
34	Presenta valor asignado a la variable C.

RUN es la instrucción del lenguaje BASIC empleada para ejecutar un programa. Debemos teclear la tal cual, sin ningún número de línea delante, pues no es una instrucción del programa.

Demos un paso más y veamos cómo puede mejorarse nuestro primer programa en BASIC. En lugar de repetir tantas veces, en varias líneas, la instrucción INPUT para recibir datos, se pueden indicar las variables una después de otra en una sola línea de INPUT. Así el programa anterior lo podríamos escribir:

```
10 INPUT A, B, C.
40 PRINT A.
50 PRINT B.
60 PRINT C.
```

Este programa es perfectamente equivalente al primero. INPUT es una instrucción que puede actuar sobre cualquier grupo de variables colocadas una después de otra.

También la instrucción PRINT se comporta del mismo modo. En lugar de repetirla tres veces, en las líneas 40, 50 y 60, se puede escribir en una sola.

```
10 INPUT A, B, C.
40 PRINT A, B, C.
```

No obstante, si ejecutamos los dos programas observaremos una cierta diferencia. En el primer caso, los tres datos aparecen en la pantalla en columna, como vimos antes, mientras que ahora aparecerán en la misma línea, algo separados entre sí. Es decir:

```
RUN
?12
?23
?34
12 23 24
```

Si en lugar de emplear comas para separar las variables A, B y C en la instrucción PRINT hubiésemos utilizado punto y coma, habríamos obtenido los datos sin separación alguna.

Estas peculiaridades de la instrucción PRINT pueden parecer enojosas, pero son las que, si se utilizan bien, nos permitirán hacer agradable y eficiente la presentación visual de los datos de un programa. Con demasiada frecuencia, nos encotramos programas muy complejos y llenos de buenas ideas, pero con una deficiente presentación visual de los resultados en la pantalla.

Como numerar las líneas

Un programa está constituido por una sucesión de líneas (líneas escritas), cada una de las cuales contiene una o más «órdenes», que se conocen como instrucciones. Cada lenguaje tiene sus propias instrucciones. Las instrucciones de un lenguaje no son nunca las mismas que las de otro, por lo que es fácil reconocer en cuál se ha escrito un programa observando las instrucciones particulares que utiliza.

Cuando veamos el listado (relación con el contenido de